Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

BACHELOR DEGREE THESIS

TITLE: Design and proof of concept of a centralized controller for Time-Sensitive Networks (TSN)

DEGREE: Bachelor Degree on Network Engineering

AUTHOR: Joan Feliu Castaño Cid

DIRECTOR: David Rincón Rivera, Anna Agustí Torra

DATE: 7 July 2018

Title: Design and proof of concept of a centralized controller for time-sensitive networks

Author: Joan Feliu Castaño Cid

Director: David Rincón Rivera, Anna Agustí Torra

Date: 7 July 2018

Overview

TSN (Time Sensitive Networking) is a set of standards created in order to deliver time-sensitive transmission of data in Ethernet networks.
This thesis describes both the current TSN standards, focusing on the central configuration aspect, and a proposal to build a centralized controller (CNC) prototype for TSN based on the study of real TSN hardware and software implementation.

This document includes in a section describing the state of the art on the TSN technology and the main projects related to centralized network control in TSN networks, a section describing all the proposed elements needed in order to create the CNC, and lastly a section describing the practical experience with real TSN-enabled hardware on the EETAC laboratory.

This project is an initiative based on the previous study of TSN trends, summarized in the first chapter of the thesis, that helped the TSN team at EETAC decide that CNC is the most interesting subject of study at this moment from the ongoing TSN-related projects.

The final goal of the thesis is to provide an overall view of the requirements on the centralized configuration in order to help new development teams to have a comprehensive starting point to continue evolving this technology into the future and integrate of the well-known Ethernet networks into the Operational Technology (OT), Internet of Things (IoT) and time-sensitive Information Technology (IT) environments.

The results from this thesis are both the setup and test of a small TSN network environment and the proposal of the design of the next step to create a real CNC will be the development of the software tools identified in this thesis.

Títol: Disseny i prova de concepte d'un controlador centralitzat per a xarxes TSN

Autor: Joan Feliu Castaño Cid

Director: David Rincón Rivera, Anna Agustí Torra

Data: 7 de Juliol de 2018

Overview

TSN (Time Sensitive Networking) és un conjunt de normes creades per oferir una transmissió de dades sensible al temps a les xarxes Ethernet. Aquesta tesi descriu tant els projectes actuals relacionats amb TSN, centrant-se en l'aspecte de la configuració central, com una proposta per construir un controlador centralitzat (CNC) per a TSN basat en l'estudi de la implementació real de maquinari i programari TSN.

Aquest document està estructurat en una secció teòrica basada en una investigació exhaustiva sobre la tecnologia TSN i els principals projectes relacionats amb el control de xarxa centralitzada en xarxes TSN, una secció que descriu tots els elements proposats necessaris per crear el CNC i, finalment, una secció que descriu les pràctiques experimentals amb el maquinari real de TSN habilitat al laboratori EETAC.

Aquest projecte és una iniciativa basada en l'estudi previ de tendències TSN, resumit en el primer capítol de la tesi, que va ajudar a l'equip de TSN a EETAC a decidir que el CNC és el tema d'estudi més interessant en aquest moment dels projectes en curs de TSN.

L'objectiu final de la tesi és proporcionar una visió general dels avenços en la configuració centralitzada per ajudar els nous equips de desenvolupament a tenir un punt de partida integral per continuar evolucionant aquesta tecnologia cap al futur i poder oferir aquesta peça central que pot provocar l'éxit de les conegudes xarxes Ethernet en els entorns de Operational Technology (OT), Internet of Things (IoT) i xarxes Information Technology (IT) sensibles al temps.

Els resultats d'aquesta tesi són tant la configuració i la prova d'un petit entorn de xarxa de TSN com la proposta d'un CNC basat en l'estudi i l'experimentació dels components necessaris. El següent pas per crear un autèntic CNC serà el desenvolupament de les eines de programari identificades en aquest TFG.

INDEX

INTRODUCTION

The Ethernet network technology, described in the standard IEEE 802.3, is one of the most used and known alternatives in local area networks, especially in Information Technology (IT) environments. The industrial transformation, known as Industry 4.0, and the spread of Internet of Things (IoT) networks in urban areas create the need of network solutions easy to deploy, and Ethernet is the most convenient option out there. Inter-vehicular and intra-vehicular communication and sensor networks are only two examples of scenarios where a cheap and well-known network technology could accelerate the development and implementation of these new trends. While Ethernet is a consolidated technology, its base characteristics are not prepared to accommodate low-latency, low-jitter traffic, as the best-effort nature of these networks was never intended to serve critical traffic and time-sensitive streams.

To evolve the Ethernet networks to be competitive in time-sensitive scenarios, specifically in TV and audiovisual production, the Audio Video Bridging (AVB) project started in 2011. With the main objective of providing the specifications that will allow time-synchronized low latency streaming services through IEEE 802 networks, the project created standards like the IEEE 802.1AS: Timing and Synchronization for Time-Sensitive Applications. The task group changed its name to the Time-Sensitive Networking Task Group (TSN) in 2012, when the members decided to broaden their scope to provide time-sensitive solutions to more sectors aside from the audiovisual media networks.

In parallel, software Defined Networking (SDN) is being developed as a new network management approach based on virtualization of networks, functions and equipment, and the centralization of control functions. SDN networks separate the data and control planes, hosting the network controllers in servers while keeping the network hardware elements as simple and reconfigurable as possible. Considering the benefits that a centralized controlled network like SDN offers, it is obvious that TSN could take advantage of the same control plane structure.

The main objective of this thesis is to study how an SDN-inspired central controller can be applied to TSN networks. At this moment there is one standard being developed in the TSN Task Group, the IEEE802.1Qcc, and multiple proposals and vendor-specific implementations. Following the leads of the previous works in this topic and investigating how existing TSN hardware works, our aim is to gather this knowledge to propose the elements and to create this centralized controller. These elements are independent of the used hardware, so using any open TSN system with the possibility of running Linux services and protocol implementations will be enough to start implementing a real CNC. Although if this thesis is based on practical experiments and tests using an specific hardware, it is mostly focused in how the different software elements are used in a real time scenario, and how to combine and expand them in order to test a real CNC prototype. The motivation behind this study is provide with a solution not only to local TSN networks, but to help the integration of TSN technology in OPC-UA industrial networks, 5G fronthaul

management or any time-sensitive environment, knowing that the centralization of controller functions is the way to go in modern networking technology.

This thesis is organized as follows:

- Chapter 1 presents the current state of TSN technology, the most promising ongoing projects related to TSN and some alternatives to deploy deterministic Ethernet networks based on non-TSN technologies.

- Chapter 2 focuses on the SDN-inspired TSN central configuration, describing both the needed technologies in detail and some existing proposals by networking hardware and software vendors. The chapter concludes with a proposal for the design of a CNC based on all knowledge gathered in this thesis, trying to set a path that can lead to the final implementation of a real TSN central controller in future works.

- Chapter 3 describes the experimentation and testing of TSN-enabled hardware on EETAC's laboratory, specifically tests with RESTCONF and LLDP. The goal of the experiments is to understand the additional elements involved in a real TSN scenario and how to implement the elements to deploy a centralized TSN network, leading to the proposal at the end of Chapter 2.

- Lastly, the Conclusions chapter summarizes the results obtained and presents the next steps advised to continue this investigation line into the near future.

Additionally, we include some annexes:

- Annex 1 serves as a tutorial on how to deploy the TSN equipment in EETAC's laboratory and includes a guide that describes the TSN demos that the equipment offers.

- Annex 2 contains information about the usage of the CURL tool, useful to test the RESTCONF server included in the TSN equipment.

- Annex 3 includes an entire YANG model, expanding the information on the 2.2.1.1 section.

- Annex 4 includes captured HTTP messages used by the RESTCONF server, expanding both the sections 2.2.1.2 and 3.2.

- Annex 5 includes information about the TSN equipment used in EETAC's laboratory and the usage of TSN standards by this equipment.

- Annex 6 includes information about TSN schedulers and network constraints.

- Annex 7 contains an explanation about how TSN can integrate with OPC UA OT networks.

CHAPTER 1. STATE OF THE ART IN TIME SENSITIVE NETWORKING (TSN)

This chapter focuses in describing the TSN technology and the related projects that are currently being developed, as well as a preliminary analysis on the benefits of applying SDN to TSN.

1.1. Time Sensitive Networking

The IEEE created the Audio Video-Bridging (AVB) Task Group [24] with the objective of developing standards that reflect the specifications of low-latency streaming and synchronized services through IEEE 802 networks. These standards allow establishing routes through the network assigning bandwidth, guaranteeing a limited latency and bounded jitter. AVB bases its operations on a system of talkers and listeners, where the listeners ask using hop-by-hop messages for a route to the talker with a certain bandwidth, latency and jitter. In parallel, Precision Time Protocol (also known as IEEE 1588 and adapted for TSN as IEEE 802.1AS) [25] is needed to synchronize the nodes with a time base with an accuracy on the order of tens or hundreds of nanoseconds.

In 2012, the AVB Task Group evolved towards the TSN Task Group [26] to cover more sectors apart from the audio-visual field: industry, automotive, manufacturing, transport and process control, aerospace, mobile networks, to name a few. TSN enables deterministic communications over legacy Ethernet networks. Thanks to the experience acquired in the development of the AVB standards, new TSN challenges are posed to solve past errors: moving from a decentralized control to a centralized architecture (more suitable for industrial applications), or a new method of traffic shaping that eliminates losses due to congestion and decreases the average delay, among others. All these features allow the convergence of massive data and control networks to a single interconnected network in real time and without any interruptions for critical tasks (control, for example). In addition to improving AVB standards, it was also proposed to design new standards in order to allow real-time communications with guaranteed low latency, transporting critical traffic without losses caused by congestion in all critical control circuits, reducing complexity and infrastructure costs by converging control and data planes, ensuring critical traffic immunity by reserving temporary slots for transmission, facilitating the design and maintenance of the network, as well as guaranteeing certain QoS of a product or service.

1.2. Main characteristics of Time Sensitive Networking (TSN)

TSN is an evolving technology currently focused on the study and development of three main topics:

1- Time synchronization in all the devices, including switches and other components between the two ends of the connection.

This can be obtained using GPS clocks in each device or distributing the clock via specific protocols, like the IEEE 1588 Precision Time Protocol (PTP). An end-to-end transparent clock is not enough (a transparent clock computes the variable delay as the PTP packets pass through the network device, improving the synchronization between master and slave clocks, being useful in networks with multiple levels of boundary clocks than can create imprecisions in synchronization), since all the devices in the network must be time-aware to have a real TSN scenario.

There is also a need of an RTOS (real time operating system) or specific hardware to solve the time delivery within the system.

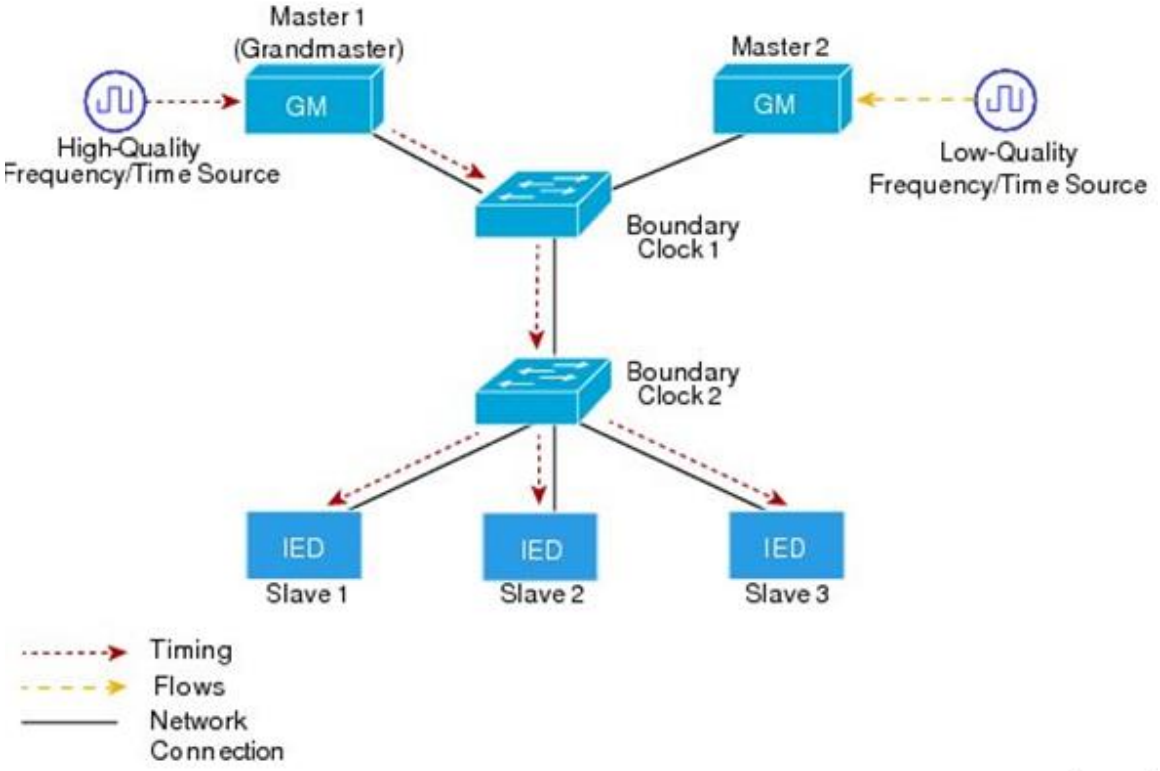


Figure 1.1 Typical PTP synchronization network (taken from [27])

2- Scheduling and traffic shaping.

There is a need for prioritizing some traffic, using high priority time slots in each time slice (IEEE 802.1Qbv time-aware scheduler) [28]. TSN allows the use of a credit-based scheduler (based on token bucket) and a time-aware scheduler, as shown in Figure 1.2.

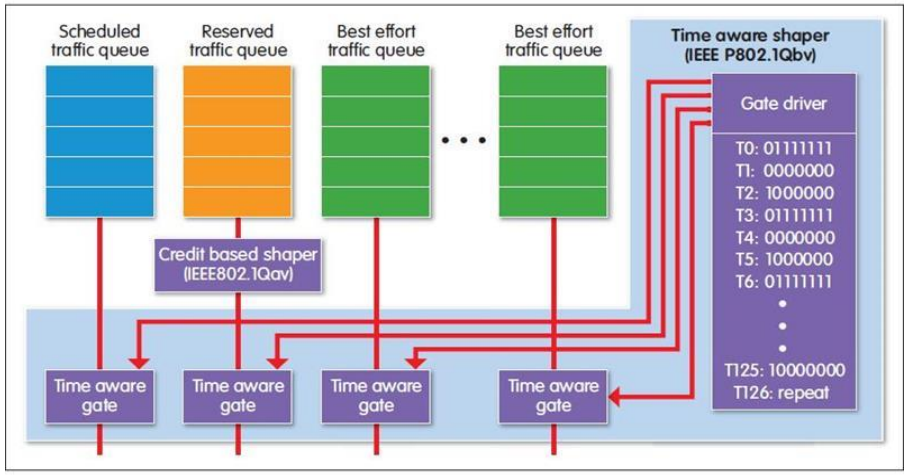


Figure 1.2 Time-Aware and Credit-Based Shapers (taken from [29])

To prevent packets from a low priority time slot to “invade” the high priority time slots, there is the need to assign guard bands before each high priority slot. Using the mechanism of frame pre-emption, low priority frames can be sent in more than one burst, pausing the sending during the high priority time slot and then resume it with the rest of the frame. This approach helps to shorten the length of the guard bands, thus avoiding a loss of bandwidth. Figure 1.3 illustrates these concepts.

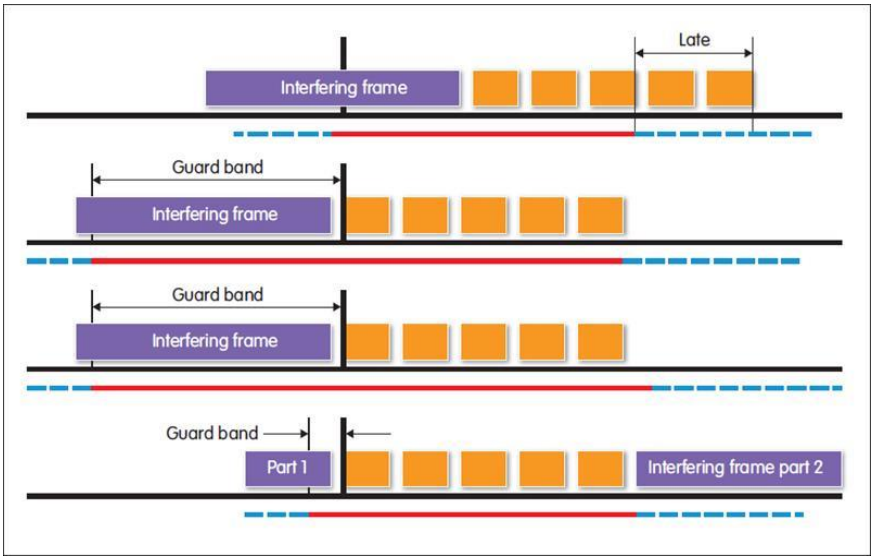


Figure 1.3 Use of frame pre-emption to maximize the usage of existing bandwidth (taken from [29])

3-Selection of communication paths, reservation and fault-tolerance.
The TSN task group is currently specifying the fault-tolerance protocol IEEE 802.1CB [32], based on the transmission of duplicated packets to provide fault tolerance based on a redundant transmission through different paths. The duplicated packets need to be detected and then eliminated to prevent transmission inconsistency.

In the currently ongoing project IEEE 802.1Qcc [30] (Stream Reservation Protocol (SRP) Enhancements & Performance Improvements), the TSN task group focuses on the definition of management interfaces and protocols to enable TSN network administration on large scale networks. Different aspects are discussed here, both with a de-centralized approach as well as a fully centralized approach that re-uses configuration concepts from software-defined networking (SDN).

1.3. TSN-related standards

To fulfill the requirements of the three main TSN priorities, the IEEE has created a set of working groups destined to study and isolate the different TSN necessities.

- Queuing and forwarding of time-sensitive streams:
  802.1Qav Credit-Based and Time-Aware Shapers, new 802.1Qbu preemption, 802.1Qbv time-aware queuing. 802.1Qch cyclic queueing, 802.1Qci input gating, and 802.1CB seamless redundancy (duplicated paths that discard the duplicated frames at the end).

- Registration and reservation of time-sensitive streams:
  802.1Qat a distributed “stream reservation protocol”, extended in new 802.1Qcc to support preemption, scheduling, centralized control, and interaction with higher layer IETF services.

- Time synchronization: IEEE Std802.1AS (based on IEEE 1588).

- Overall system architecture: IEEE Std802.1BA “audio video systems”, P802.1CM fronthaul systems for mobile (“radio over Ethernet”).

At the same time, there are several ongoing projects studying how to further evolve TSN. The next list briefly describes the current state (as of June 2018) of the ongoing TSN projects described in the IEEE website:

- 802.1CS – Link-local Registration Protocol (last updated January 2018):
  Link-local Registration Protocol (LRP) is focused on helping the creation of application protocols that distribute information between network devices. The objective is replicate a registration database and its changes from one node to another, in a point-to-point link. The main challenge is overcoming the scalability problems created by MRP, and the capability to use databases on the order of 1 Mbyte efficiently.

- 802.1AX-Rev – Link Aggregation Revision (last updated November 2017):
  Link aggregation provides protocols, procedures and managed objects that allow for parallel instances of point-to-point links to be aggregated together, such that a MAC client can consider them a single link with higher bandwidth than the individual links.

- 802.1AS-Rev – Timing and Synchronization for Time-Sensitive Applications (last updated November 2016):
  This standard specifies the protocol and procedures to ensure the synchronization requirements, jitter and wander are met for time aware applications and network components that work in time-sensitive networks. The latest proposals in this revision are using 1588-rev data sets as a foundation for 1AS-rev data sets, using 1588 (PTP) YANG module as a base for 1AS-rev module, and eliminating the problems caused by networks shared between products that use 802.1AS and other 1588 profiles using a profiling system.

- 802.1Qcc – Stream Reservation Protocol (SRP) Enhancements and Performance Improvements (last updated January 2018):
  This project is further described on Chapter 2 from this document.

- 802.1Qcj – Automatic Attachment to Provider Backbone Bridging (PBB) services (last updated February 2016):
  This standard specifies the protocols, procedures and management objects for auto-attachment of network devices to Provider Backbone service instances using values within LLDP.

- 802.1CM – Time-Sensitive Networking for Fronthaul (last updated January 2018):
  This standard defines profiles that select features, options, configurations, defaults, protocols and procedures of bridges, stations and LANs that are necessary to build networks that are capable of transporting fronthaul streams, which are time sensitive.
  The standard proposals are based on separating the computing, expensive part of a wireless base station from the “dumb” and cheap antennas, inspired by Common Public Radio Interface (CPRI) [34], a standard for communications between radio equipment controllers and radio equipment. The communication between the two parts needs strict timing requirements, and TSN over Ethernet and its many defined protocols is considered The last reports describe three different traffic classes for fronthaul traffic differentiating maximum delay and packet loss ratio.

  Some EETAC’s researchers, including the author of this thesis, presented recently a study [5] about the implementation of TSN central configuration based in SDN networks, with the aim of helping the deployment of 5G environments enabled by Cloud Ran (C-RAN).

- 802.1Qcp – Bridges and Bridged Networks Amendment: YANG Data Model (last updated September 2016):
  Defines a YANG data model that allows configuration and status reporting for bridges and bridge components including MAC Bridges, Two-Port MAC Relays (TPMRs), VLAN Bridges, and Provider Bridges with the capabilities currently specified in this standard. This YANG module is used by the Jetconf server (*module ieee802-dot1q-bridge*) in our laboratory scenario.

- 802.1Qcr – Bridges and Bridged Networks Amendment: Asynchronous Traffic Shaping (last updated January 2018):
  This standard specifies procedures and managed objects for a bridge to perform asynchronous traffic shaping over full-duplex links, with constant bit data rates. This project does not relay on synchronous communication, providing independence from clock synchronization and higher link utilization than mechanisms that uses that synchronization. It describes both the YANG module and scheduling algorithms that provide bounded networks delays without the use of protocols like PTP.

- 802.1ABcu – LLDP YANG Data Model (last updated December 2017) and -802.1Qcw – YANG Data Models for Scheduled Traffic, Frame Preemption, and Per-Stream Filtering and Policing (last updated December 2017):
  Provides a YANG model that allows configuration and status reporting for bridges with regards to topology discovery (LLDP), scheduling, frame preemption and per-stream filtering and policing. It is an important piece to the centralization of control in TSN networks.

For more information on TSN, read [31], a bachelor thesis that summarizes all the TSN existing protocols.

1.4. SDN for Real Time Ethernet (RTE)

This section analyzes the benefits of applying SDN for Real-Time Ethernet, and also mentions a couple of works related to SDN and PTP.

1.4.1. Benefits of SDN for RTE

RTE allows the use of cost effective, widespread and high-bandwidth Ethernet technology in industrial environment, as described in [1]. Typical RTE deployments in the past have been configured once to run without re-configuration for years. However, in the future RTE networks will need to be more flexible.

SDN is a technology that provides a great range of freedom to flexibly and centrally reconfigure the network on-demand. The basic idea of SDN is to control network flows through a centralized intelligent controller with “dumb” forwarding devices in the data plane of the network. An interesting idea is replacing the switches/hubs of real-time Ethernet solutions with SDN-capable

switches and extend RTE protocols by providing additional features that the use of SDN controllers and switches make possible.

RTEs usually define a time-slotted channel for ensuring a share of the channel for each user. The performance of RTE can be described by cycle times and data rate. Cycle time is the duration of one transmission cycle, and is relevant for applications that need to transmit often small amounts of data. The data rate is the maximum rate of data that can be transmitted over a single link, and is important for applications that want to transmit large amounts of data. RTE solutions have two operating principles: time schedule, where the devices use a pre-defined time network schedule that defines which device can transmit at each time, and polling, where a single master server queries all clients according to its internal schedule, being the clients only able to transmit data in response to a server's query.

One of SDN's main capabilities is the fine-grained control of data flows in the network. Therefore, RTE features like broadcasting, real multicasting, concurrency, arbitrary topologies, redundancy and multipath routing will be as realizable using SDN as using more traditional networking approaches. SDN will potentially allow for a more efficient solution. However, one key limitation needs to be pointed out: standard SDN devices currently do not support frame forwarding at precise moments in time and, thus, do not naturally support time-scheduled protocols. However, adding the notion of time does not conceptually contradict the use of SDN and is thus rather an implementation issue.

Advantages gained applying SDN over RTE solutions:

- Central configuration: A key feature of SDN, allows adaptive device settings, communication paths and schedules. This whole document is centered on this idea.

- Standardization: OpenFlow, the most widely used SDN protocol, and newer protocols like RESTCONF and NETCONF, define a set of functionalities that all network devices must fulfill, and a standard interface to access the functions.

- Global network information: SDN devices collect useful statistics about packet loss ratio and average bandwidth usage, being important in order to have a network state overview.

- Central addition and removal of network nodes: Based on OpenFlow or NETCONF, nodes can be dynamically added or removed from the real-time network, and recombined to fulfill different tasks.

- Arbitrary topology: Due to the central configuration, loops do not pose a problem for SDN, having an advantage respect traditional networks that use methods like the spanning tree protocol (STP).

- Fast reroute and failover: SDN network can use additional backup routes transparently, for time-scheduled protocols the schedules have to be

adapted to avoid congestion in the backup paths. For zero-loss or zero-time failovers, flows can be duplicated.

- Multiple simultaneous communication paths: Aside from backup, additional paths or duplication of redundant flows can be used to increase available bandwidth.

- Multiple networks over one infrastructure: An SDN approach to RTE networks could enable the operation of multiple RTE networks over a single physical infrastructure. Such a setup may require time synchronization between devices from the multiple networks. This feature could be useful for polling-based and time-based protocols.

- Isolation of faulty nodes: Using SDN faulty network nodes can be easily disconnected from the network. The SDN/RTE controller needs to detect faulty nodes using, for example, frame counting, and even use a selective node isolation allowing only correct frames to pass and blocking incorrect or late frames.

- Dynamic load balancing: This technique allows the dynamic change of communication paths or the simultaneous use of multiple paths between a sender and a receiver to avoid congestion.

- Efficient multicasting: When sending multicast traffic using SDN, avoiding links where there is no subscribers to this traffic helps saving bandwidth and allowing real-time protocols to use multiple parallel communication flows. And protocols that only allow one sender in the network at a time, would benefit from a security point of view as unsubscribed nodes will not receive any multicast frames.

The implementation of RTE using current SDN technology presents some small issues. While we do not know any conceptual reason which would prevent the support of time schedules in SDN switches, we are not aware of any standard SDN switches which support schedules. Additionally, when low cycle times are required, the performance guarantees depend on the achievable forwarding latency and jitter of SDN switches. It is necessary to measure the performance of SDN switches and compare it to current Ethernet switches and hubs used for RTE. Finally, SDN in general does not depend on the use of Ethernet compatible frames, however current OpenFlow compatible switches do pose that requirement.

There are also some disadvantages introduced by SDN. One key disadvantage of SDN is the need for a controller. Such a controller is a single point of failure and a controller failure would disable further central network configurations. However, this shortcoming prevents only use cases in which it is necessary to reconfigure the network while deterministic traffic is transferred over the network. In all other cases, the guaranteed performance would not be affected even if the SDN controller failed; only reconfiguration would be disabled.

The conclusion is that the development of a software-defined real-time Ethernet is a highly promising endeavor that needs validation in a test network.

1.4.2. Reverse PTP

An interesting technology that implements real-time capabilities to SDN networks is Reverse PTP [33]. Reverse PTP is a clock synchronization scheme for software-defined networks, based on the Precision Time Protocol (PTP) but conceptually reversed. In Reverse PTP all nodes in a network distribute timing information to a single SDN node, the controller, a software-based node. The node tracks the state of all the clocks in the networks and performs all computations and the network devices only send periodically their current time. Reverse PTP offers a centralized alternative to the fully distributed behavior of PTP, making it more in line with SDN standards and offering the benefits of a time synchronized network environment.

1.4.3. Fully Synchronous SDN (FSS)

The Fully Synchronous SDN (FSS) architecture combines SDN, SyncE [36] (FSS/SyncE) and PTP (FSS/PTP), as detailed in [35], in order to create a multitenant network which provides a synchronization service to multiple groups of users. FSS extends in a standard-compliant way the control plane capabilities of OpenFlow-based SDN to include the management of the synchronization of core and access networks in the 5G scenario, thanks to its slice-awareness and multi-tenancy capability. The proposed architecture [36] integrates SDN, SyncE and PTP, and is also compatible with legacy nonsynchronized equipment, thus easing equipment upgrade in operation networks and reducing costs.

1.5. Thesis goals

Knowing how the TSN technology is evolving and the most promising trends in IT, OT and IoT network development, this thesis s focused in the study of SDN-inspired centralized configuration for TSN networks. The main goal of the thesis is the creation of a proposal for a centralized controller for TSN networks, based in researching how this technology is shaping, both investigating open standards and vendor-specific solutions, and testing real TSN equipment in the laboratory. This proposal will be focused mainly in the software part of a TSN network, describing protocols and existing implementations, trying to find the most optimal and complete solution.

CHAPTER 2. SDN-BASED MANAGEMENT OF TSN

This chapter describes the main proposals to develop a centralized control plane for TSN networks, the necessary elements to this development effort and some proposals created by TSN hardware and software vendors.

2.1. Architecture of the centralized management approach

At this moment, the consensus in the IEEE TSN working groups is that a centralized approach to the control for TSN elements is better than the hop-by-hop method used in AVB [24]. This section presents the main topics related to the creation of a TSN CNC (Central Network Controller), based on the last developments in the IEEE802 TSN task group.

Figure 2.1 represents the proposed structure that the majority of vendors and researchers follow in order to implement the TSN central configuration.

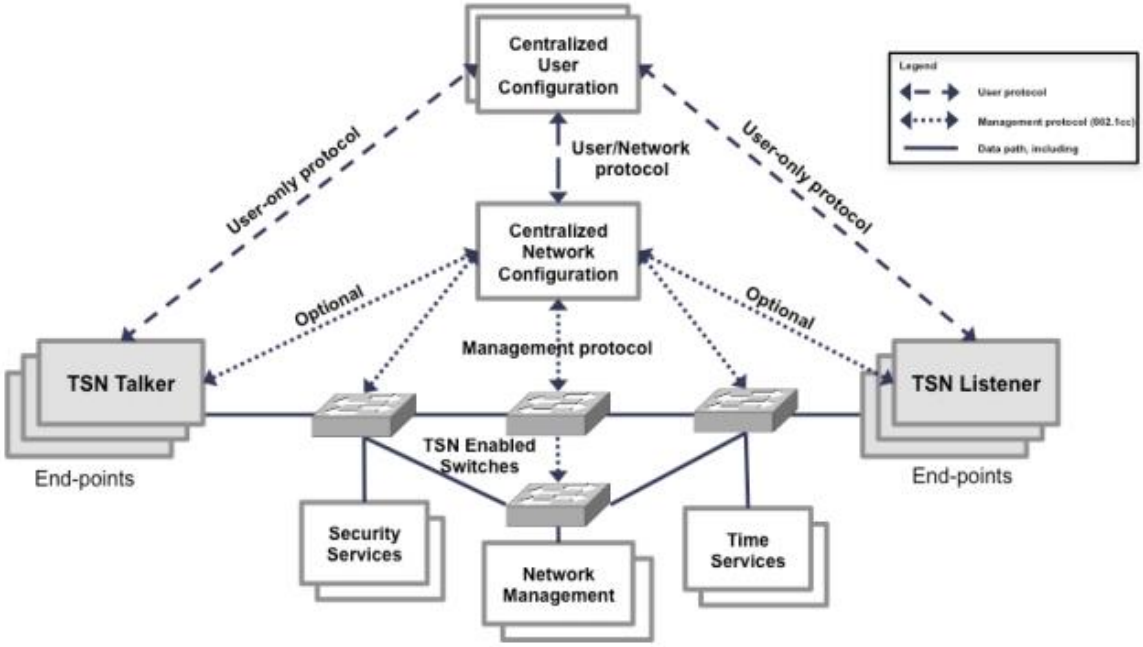


Figure 2.1 A model for TSN Central configuration (taken from [6])

The CNC and CUC are elements present in the centralized version of TSN control, using YANG Models ("Yet Another Next Generation") data models to describe properties of a TSN machine and IETF NETCONF/RESTCONF protocols to communicate between them and the bridges.

The two main elements are:

- Central network controller (CNC): For TSN, the CNC acts as a proxy network (the TSN bridges and their interconnections) to the applications that require deterministic communication. Usually the CNC application is

provided by the vendor of the TSN bridges. It has two primary responsibilities. First, it is responsible for determining routes and scheduling the TSN flows through the bridged network. This operation includes calculating the schedule that the TSN hardware will use. Second, it is responsible for configuring the TSN bridges.

The hardware housing the CNC application is not defined. Some vendors offer a pre-configured virtual machine for an easy deployment.

- Centralized user configuration (CUC): An application that communicates with the CNC and the end devices. The CUC makes requests to the CNC for deterministic communication (TSN flows) with specific requirements for those flows, that are determined by the needs of the end-point applications requesting a TSN-enabled connection. The CUC is an application that is vendor specific, typically the vendor of the TSN end devices will supply a CUC for those end devices. The CUC connects to the CNC using REST APIs.

2.2. Central management elements

In this section we will discuss the software elements needed to implement a TSN centralized controller. The existing elements section describes elements that are already finished or are in an advanced prototype version, and can be implemented. All this elements are working in our TSN scenario. The implementation proposals section describes proposals of CNC elements or CNC integral implementations by some TSN hardware/software vendors. The conclusions and proposals sections discuss the next steps needed to evolve the CNC technology, both summarizing the progress in the implementation of the needed elements and proposing an integration of these elements.

2.2.1 Existing CNC elements

The following section explains the existing elements that are considered key elements by the IEEE TSN task force in order to define the structure for a future CNC. We will now present the concepts related to the YANG data models, the RESTCONF protocol and the LLDP network discovery protocol.

2.2.1.1 The YANG data model

Modern network equipment use the RESTCONF/NETCONF protocols to obtain configuration and capabilities information from network devices, and to change that configuration. These protocols work with YANG [7], a modeling language used for data definition. The data models are yet to be finished for all TSN operations.

During our lab experience with the SOC-e hardware, described in Annexes 1, 3, 4 and 5, the SOC-e developers confirmed that the YANG data model for

IEEE 802.1Qbv used on the hardware is the one created for testing purposes by National Instruments [21].

We will now provide more details about the data model, both for a better understanding of how the language works and how implements the TSN configuration.

All YANG models use an XML tree format structure. Depending on the RESTCONF/NETCONF implementation, the YANG models can be encoded using JSON or XML format. The XML tree structure needs a hierarchy in the definition of the elements. An example of how the hierarchy works:

```
module example-sports {
namespace "http://example.com/example-sports";
  prefix sports;
import ietf-yang-types { prefix yang; }
typedef season {
    type string;
    description
      "The name of a sports season, including the type and the year, e.g,
       'Champions League 2014/2015'.";   }
 container sports {
    config true;
      list person {
      key name;
      leaf name { type string; }
      leaf birthday { type yang:date-and-time; mandatory true; }  }
  list team {
      key name;
      leaf name { type string; }
      list player {
        key "name season";
        unique number;
        leaf name { type leafref { path "/sports/person/name"; }  }
        leaf season { type season; }
        leaf number { type uint16; mandatory true; }
        leaf scores { type uint16; default 0; }
      }}}}
```

This example shows a data model for the module example-sports. The module declares a namespace and a prefix and imports the type library module “ietf-yang-types” before defining the type “season”. It then defines a container and two lists inside it, each one with its elements (leaf). Note that the “name” leaf in the team list relates to the same leaf in the “person” list. Let’s study some interesting fields in the 802.1Qbv YANG model.

```
module ieee802-dot1q-sched {
  namespace "urn:ieee:std:802.1Q:yang:ieee802-dot1q-sched";
  prefix "sched";

  import ietf-yang-types { prefix "yang"; }
  import ieee802-dot1q-types { prefix "dot1q-types"; }
  import ietf-interfaces { prefix "if"; }
  import ieee802-dot1q-bridge { prefix "dot1q"; }
```

The module name is "*ieee802-dot1q-sched"*, name related to the IEEE 802.1q (VLAN) and to the word Scheduling.
The *prefix* is the string used when the module is imported. It must be unique within the module/submodule. The keyword *import* means that the module references other external modules, while *include* means that the module references other internal modules (submodules within the same module). So this module imports the IETF models for YANG types and interfaces, and also imports other 802.1q modules, the one for common types and the one for bridge configuration.

```
description
"This module provides for management of IEEE Std 802.1Q Bridges that
support Scheduled Traffic Enhancements.

This experimental YANG module is an individual  contribution,and does not
represent a formally sanctioned    YANG module of IEEE.
Therefore, this YANG module will change in incompatible ways
from its current revision to the formally published YANG
module for IEEE Std 802.1Qbv-2015.";
```

This is a textual *description* useful to understand the use of the module.

```
feature scheduled-traffic {
  description
  "Each Port supports the Enhancements for Scheduled   Traffic.";
  reference
  "IEEE Std 802.1Qbv-2015"; }
```

A *feature* section represents a conditional portion of the model, only enabled if the hardware supports it. Scheduled traffic will be used in our hardware because it supports the function, but thanks to that option the modeler can include it without worrying for the model to not being compatible with other equipment.

```
 grouping ptp-timestamp {
  description
"This grouping specifies a PTP timestamp, represented as a 48-bit
unsigned integer number of   seconds and a 32-bit unsigned integer number
of  nanoseconds.";
  reference
 "IEEE Std 802.1AS-2011: 6.3.3.4";
  leaf seconds {
  type uint64;
      description
        "This is the integer portion of the timestamp in
        units of seconds. The upper 16 bits are always zero.";
    }
    leaf fractional-seconds {
      type uint64;
      description
   "This is the fractional portion of the timestamp
    in units of nanoseconds. This value is always
    less than 10^9.";}
```

A *grouping* is a set of nodes that can be reused by other nodes, can be instantiated by the “uses” keyword. The type Uint6, represents a 64-bit unsigned integer, so a positive or null value. The *description* in each leaf make them self-explanatory. The *reference* keyword is an string that relates the *grouping* to a document that provides information on the PTP timestamping.

```
 augment "/if:interfaces/if:interface/dot1q:bridge-port" {
    if-feature scheduled-traffic;
    description
      "Augment the dot1q bridge-port with Scheduled Traffic
      configuration.";
```

The *augment* statement adds more elements to the external module associated to its definition.

The *if-feature* makes the parent statement (in this case *augment*) conditional depending in if the feature is supported or not, in this case scheduled traffic.

The rest of the document discusses elements inside this *augment* block.

```
list max-sdu-table {
     key "traffic-class";
     description
"A list containing a set of max SDU parameters, one for each
traffic class. All writable objects in this table must be
persistent over power up restart/reboot.";
     reference
       "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9, 12.29.1";

    leaf traffic-class {
       type dot1q-types:traffic-class-type;
       description
         "Traffic class";
     }
     leaf queue-max-sdu {
       type uint32;
       default "0";
       description
         "The value of the queueMaxSDU parameter for the traffic
         class. A value of 0 is interpreted as the max SDU size
         supported by the underlying MAC. The value must be
         retained across reinitializations of the management
         system.";
       reference
         "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9, 12.29.1.1.1";
     }
     leaf transmission-overrun {
       type yang:counter64;
       default "0";
       description
         "A counter of transmission overrun events, where
         a PDU is still being transmitted by a MAC at the
         time when the transmission gate for the queue closed.";
       reference
         "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9, 12.29.1.1.2";
     } }
```

A *list* contains a group of *leaf* elements related, in this case related to the SDU. SDU means service data unit, a unit of data that has been passed down from an OSI layer to a lower layer, the payload viewed by the protocol that is sending the data.

The *key* statement, which must be present if the list represents configuration, and may be present otherwise, takes as an argument a string that specifies a space-separated list of *leaf* identifiers of this list. A *leaf* identifier must not appear more than once in the key, in this case the *key* is the traffic class.

```
  container gate-parameters {
      description
"A list that contains the per-port manageable parameters for traffic
scheduling. For a given Port, an entry in the table exists. All writable
objects in this table must be persistent over power up restart/reboot.";
      reference
        "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9, 12.29.1";


       key "index";
        description
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9.4.19, 12.29.1.2";

        leaf index {
          type uint32;
          description
"This index is provided in order to provide a unique key per list entry.
The value of index for each entry shall be unique(but not necessarily
contiguous).";
        }

        uses gate-control-entry;
      }
      container admin-cycle-time {
```

A *container* defines an interior data node in the schema tree. The container does not have a value, but the definition of child nodes.

```
     container admin-base-time {
       reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.1, 12.29.1";
        uses ptp-timestamp;
      }
      container oper-base-time {
       reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.18, 12.29.1";
        uses ptp-timestamp;
      }
```

Certain elements have both an administrative and an operational value. The administrative value represents the value chosen by the engineer, and must be persistent even if the equipment is rebooted. The operational values represent

how is configured the value at a moment, given that the overall network configuration can change a value from its administrative form.

The full YANG model for the 802.1Qbv can be found on Annex 3.

2.2.1.2. RESTCONF

The protocol used to configure our laboratory hardware, and most of the modern network equipment, is RESTCONF [8], a protocol based on the most used Network Configuration protocol, NETCONF [38]. This section is based on the 18th version release of the protocol, being the latest and most complete one.

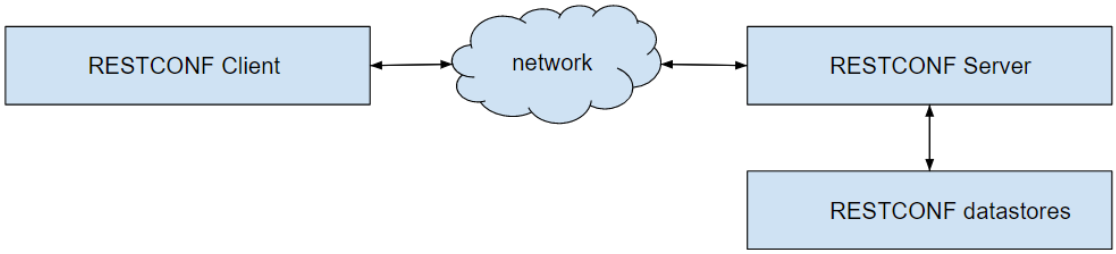


Figure 2.2 RESTCONF basic structure

The difference between the two protocols is that RESTCONF is a RESTful protocol, using the REST architecture style shared between a lot of web applications, and it has some of its characteristics, the most important of them is being sessionless.

RESTCONF is a HTTP-based protocol that provides an interface for accessing YANG data (both in XML or JSON media types), using the datastore system defined in NETCONF. Its creation responds to the need for an standard mechanism for allowing Web applications to access and configure a network device, defining a set of CRUD (Create, Read, Update and Delete) operations to work with the datastores and a protocol to invoke these operations. It needs to be compatible with NETCONF without mirroring the full functionalities, as the two protocols can share the same datastore, being two alternatives in the configuration of a network device if needed. The first version of RESTCONF was published as RFC 8040 [8].

RESTCONF uses HTTP, so the security mechanisms used by HTTP, such as Transport Layer Security are needed, due to the sensitive nature of the information shared. Using X.509 certificates and server/client authentication (HTTPS) is a must on every RESTCONF implementation.

For a better understanding of RESTCONF, the following sections describe its main characteristics: the usage a datamodel-driven API that has a defined set of resources and operations and the URI structure to access these resources and operations.

2.2.1.2.1. Datamodel-driven API

RESTCONF combines the simplicity of HTTP with the predictability of a schema-driven API. Knowing what are the useful YANG modules by a server, a client can know the structure for requests and responses for all managed resources, the server can support even the fully retrieval of YANG modules. Every RESTCONF server needs to have a module dedicated on listing all the supported modules. This module is called “ietf-yang-library”.

2.2.1.2.2. Resources

RESTCONF operates on a hierarchy of resources representing manageable components, starting with the top-level API resource. A resource can be defined as a collection of data and the methods allowed on that data.

To access the resources, HTTP CRUD methods and a set of URIs are needed. The first step is discovering where the root URI for of the RESTCONF API is located:

```
The client might send the following:

GET /.well-known/host-meta HTTP/1.1
Host: example.com
Accept: application/xrd+xml

The server might respond as follows:

HTTP/1.1 200 OK
Content-Type: application/xrd+xml
Content-Length: nnn
      <XRD xmlns='http://docs.oasis-open.org/ns/xri/xrd-1.0'>
<Link rel='restconf' href='/restconf'/>
</XRD>
```

When constructing the URI for any request, this value (/restconf/) must be used at the start of the string. For example, in the RESTCONF server used on the network equipment in our lab, the root is: 192.168.4.65:8443/restconf/. Note that on any endpoint (host:port) only one RESTCONF server can be operative due to the root resource discovery mechanism.

After discovering the root, the client can explore the API resource, that contains the RESTCONF root resource. The root resource has the following child resources: data, operations and yang-library-version.

- Data: Represents the combined configuration and state data resources that can be accessed by a client, an abstraction of the datastore implementation. It cannot be created or deleted by the client.

  To prevent collision by other RESTCONF or NETCONF server on the same datastore, two mechanisms are provided: a last-modified timestamp and an entity-tag for the datastore resource in order to allow

matching operations.The datastore resource contains descendant data nodes, that must implement too the timestamp and the entity-tag (only in configuration resources). During any operation that can modify a datastore element, these values must be checked by the petition.

```
GET /restconf/data/example-jukebox:jukebox/library\
 ?content=nonconfig HTTP/1.1
 Host: example.com
Accept: application/yang-data+xml

The server might respond as follows:
      HTTP/1.1 200 OK
 Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+xml
      <library xmlns="https://example.com/ns/example-jukebox">
<artist-count>42</artist-count>
<album-count>59</album-count>
<song-count>374</song-count>
 </library>
```

To identify a resource, an URI path expression is used in RESTCONF, being one of the main differences with NETCONF. An example:

```
URL: /restconf/data/example-top:top/list1=key1,key2,key3/\list2=key4,key5/X
```

If the leaf element is a list, there is the need to specify the keys that define the list.The ":" symbol is used when a node in the path is defined in a module different to the parent. Reserved characters need to be encoded using the percent encoding.

- Operations: A container that provides access to the operations supported by the server. Any operations defined in the YANG modules advertised by the server must be a child node of the operations resource.

```
The client might send the following:
      GET /top/restconf/operations HTTP/1.1
Host: example.com
Accept: application/yang-data+json

The server might respond as follows:
      HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
 Last-Modified: Thu, 26 Jan 2017 16:00:14 GMT
 Content-Type: application/yang-data+json
      { "operations" : { "example-jukebox:play" : [null] } }
```

The operation resources can be defined as "rpc" (Remote Procedure Call) [39] that are general for the RESTCONF server or datamodel-specific "action".

The RPC operations are invoked as:
POST {+restconf}/operations/<operation>

```
POST /restconf/operations/module-A:reset HTTP/1.1
Server: example.com
```

The actions are invoked as:
POST {+restconf}/data/<data-resource-identifier>/<action>

```
POST /restconf/data/module-A:interfaces/reset-all HTTP/1.1
Server: example.co
```

Some RPC or Actions have an input section that must be sent to specify some parameters. The next example shows how to reset a network interface:

```
POST /restconf/data/example-actions:interfaces/\
interface=eth0/reset HTTP/1.1
Host: example.com
Content-Type: application/yang-data+json
      { "example-actions:input" : {
 "delay" : 600
 }
  }
The server might respond as follows:
      HTTP/1.1 204 No Content
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
```

The reset action is tied to the interface leaf, and the delay must be sent as input because the reset operation has the delay variable defined.

Some operations can have an output aside from the server confirmation to add further info on the operation, in some cases if the operation ends with an error (400 bad request, 500 server error…) an specific message about the cause for the error can be sent. The status codes both for error and success messages are the HTTP standard ones, mapped to the NETCONF error tags.

- Yang-library-version: This mandatory leaf identifies the revision date for the YANG module implemented.

```
Example:
      GET /restconf/yang-library-version HTTP/1.1
 Host: example.com
 Accept: application/yang-data+xml

The server might respond as follows:
HTTP/1.1 200 OK
Date: Thu, 26 Jan 2017 20:56:30 GMT
Server: example-server
Cache-Control: no-cache
Content-Type: application/yang-data+xml
```

```
      <yang-library-version
xmlns="urn:ietf:params:xml:ns:yang:ietf-restconf">\        2016-06-21\
</yang-library-version>
```

2.2.1.2.3. RESTCONF methods

Table 1 compares the methods used by RESTCONF to their NETCONF counterparts:

OPTIONS	none
HEAD	<get-config>, <get>
GET	<get-config>,<get>
POST	<edit-config>(nc:operation="create")
POST	invoke an RPC operation
PUT	<copy-config> (PUT on datastore)
PUT	<edit-config> (nc:operation="create/replace")
PATCH	<edit-config> (nc:operation depends on PATCH content)
DELETE	<edit-config> (nc:operation="delete")

Table 1 RESTCONF/NETCONF method comparison

As the two protocols are compatible, there is a specific mapping between operations, while the resource path needs to be converted by the server to the YANG instance identifier. The methods used by RESTCONF are not explained in this document, as they are an HTTP specification and not a RESTCONF specific feature. An interesting note is that while PUT create or modifies the resource depending if it exists or not, PATCH only can modify, returning an error if the desired resource cannot be found.

2.2.1.2.4. Request URI Structure

The basic structure for writing an URI for an RESTCONF request is as follows:

```
Method /restconf/path resource?query
```

- Method: An HTTP method from the table described in the RESTCONF methods section
- “restconf”: The root of the RESTCONF API, discovered getting the root resource. Usually it has the “restconf” value.
- path: the path to the desired resource.
- query: defined parameters that define constraints to the request. Further explained in the query parameters section.

2.2.1.2.5. Query parameters

RESTCONF allow the use of a set of pre-defined queries to help customize the petitions. The query will be always written at the end of the URL, preceded by the “?” symbol.

```
GET /restconf/data/example-jukebox:jukebox?depth=1 HTTP/1.1
```

In the example the depth=1 query determines that the result will only show one level of child resources, so it will only show the target resource itself. Increasing the value allow the response to show the resource tree in different depths.

```
GET/restconf/data/example-events:events?\ content=config HTTP/1.1
```

This other query determines that only the configuration content in the resource must be returned.

```
POST/restconf/data/example-jukebox:jukebox/\
playlist=Foo-One?insert=first HTTP/1.1
```

The insert=first query is used to determine that the new leaf instance will be placed first in the list.

```
filter = /event/event-class='fault'
 GET /streams/NETCONF?filter=%2Fevent%2Fevent-class%3D'fault'
```

The last example shows a filter and how to implement it in the GET petition. Note the percent encoding used to write the special characters. Some of these filters are optional, to know which filters the RESTCONF server implements, use the next petition:

```
 GET          /restconf/data/ietf-restconf-monitoring:restconf-state/\
capabilities HTTP/1.1
```

2.2.1.3. Network Topology Discovery

In order to compute the time schedules, the CNC needs an overall view on all the elements on the network and how they are connected. Following the SDN inspiration, an interesting idea is to use LLDP (Link Layer Discovery Protocol) [20] [57], a protocol compatible with most networking hardware. LLDP allows network devices to announce their capabilities, using periodic messages that each network interface sends, being these messages only transmitted in the Layer 2 domain. The LLDP information is stored in a MIB (Management Information Base) [55]. A MIB is a database used for managing the entities in a communication network.

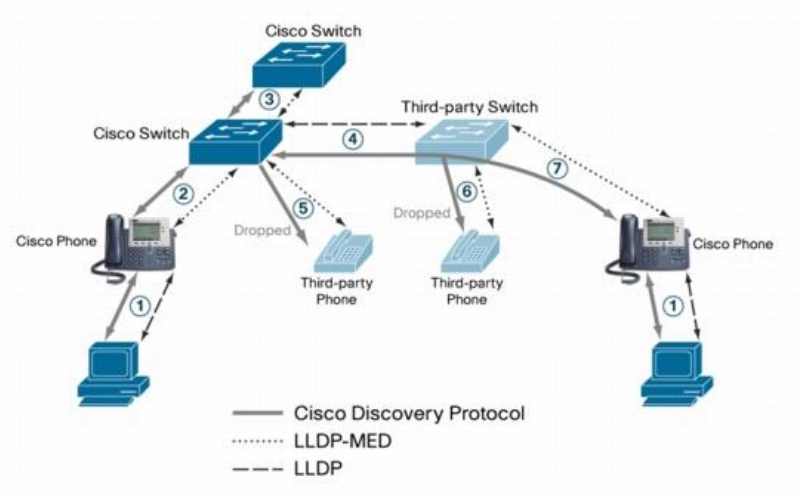


Figure 2.3 LLDP and CDP (Cisco) message propagation (taken from [9])

Being compatible with Ethernet, it is a fair assumption that installing a LLDP Agent on TSN devices does not have any compatibility issue. The CNC can use all the LLDP information to create a topology model using a script, and then send that information to the CUC that can implement some graphical tool to draw the topology.

A popular implementation of LLDP is lldpd [10], as described in more detail in Section 3.3.

Another way to know the network topology is using the RESTCONF protocol for that purpose. The LLDP MIB can be translated to a YANG model, and included in a read-only datastore. This way, a RESTCONF or NETCONF operation can be used to retrieve this information.

While the YANG model seems like the better option considering that the RESTCONF protocol is already a central piece of the CNC build, it creates a new need: thinking a way to transforms the standard MIB into a topology YANG model.

RFC 6643 [11] defines how to transform a standard switch MIB into a YANG model, defining constraints and guidelines aimed to help in the process. The next table shows an example of an ifTable (interface table) from a MIB transformed into a YANG module container.

```
container ifTable {
       description
"A list of interface entries.  The number of entries is given by the
value of ifNumber.";

       smiv2:oid "1.3.6.1.2.1.2.2";

       list ifEntry {
         key "ifIndex";
         description
"An entry containing management information applicable to aparticular
interface.";
```

```
         smiv2:oid "1.3.6.1.2.1.2.2.1";

         leaf ifIndex {
           type if-mib:InterfaceIndex;
           description
"A unique value, greater than zero, for each interface.  It
is recommended that values are assigned contiguously starting from
1.The value for each interface sub-layermust remain constant at least
from one re-initialization ofthe entity's network management system to
the next reinitialization.";

           smiv2:max-access "read-only";
           smiv2:oid "1.3.6.1.2.1.2.2.1.1";
         }
```

To understand how the transformation process works, let's take a look at the same object, ifTable from a LLDP MIB.

```
ifTable OBJECT-TYPE
    SYNTAX      SEQUENCE OF IfEntry
    MAX-ACCESS  not-accessible
    STATUS      current
    DESCRIPTION
"A list of interface entries.  The number of entries is given by the
value of ifNumber."
    ::= { interfaces 2 }

ifEntry OBJECT-TYPE
    SYNTAX      IfEntry
    MAX-ACCESS  not-accessible
    STATUS      current

    DESCRIPTION
"An entry containing management information applicable to a particular
interface."
    INDEX   { ifIndex }
    ::= { ifTable 1 }
```

Note that the *list* component of YANG modules is described as a *sequence* on the MIB, and that the MAX-ACCESS field is defined as "not-accessible". This means that the object is classified as an auxiliary object and should be defined within the conceptual row in a table.

A proposal for a YANG model to describe LLDP exists, based in the features of the LLDP version 2 MIB [12]. It contains the definitions for MIB components and LLDP agent capabilities. It is interesting to observe the first lines from the model, that describes the LLPD agent configuration:

```
feature lldp-rx {
    description
      "This LLDP agent supports receive (Rx) mode or
      transmit/receive (TxRx) mode.

Use of if-feature in this module specifies conformance
in a manner analogous to Table 11-1 of IEEE Std 802.1AB-2016(MIB
object groups and operating mode applicability) and the corresponding
```

```
conformance group in the MIB (11.5).";


reference
      "IEEE Std 802.1AB-2016: 11.2, 11.5";
  }

feature lldp-tx {
    description
    "This LLDP agent supports transmit (Tx) mode or
    transmit/receive (TxRx) mode.

Use of if-feature in this module specifies conformance in a manner
analogous to Table 11-1 of IEEE Std 802.1AB-2016(MIB object groups and
operating mode applicability) and       the corresponding conformance
group in the MIB (11.5).";

    reference
    "IEEE Std 802.1AB-2016: 11.2, 11.5";
  }
```

The Section 3.3 from this document describes a working implementation of LLDP network topology discovery and storage using RESTCONF.

2.2.2. Implementation proposals

The next section describes CNC implementation proposals by different parties, both in configuration methods and in required CNC elements. As of today there is no standard defined for CNC, but it is interesting to see how some of the best networking companies are trying to work with this technology.

2.2.2.1. Cisco's implementation

The next section is based on Cisco's work on the CNC implementation for their hardware, specifically the IE-4000 Industrial Router [13].

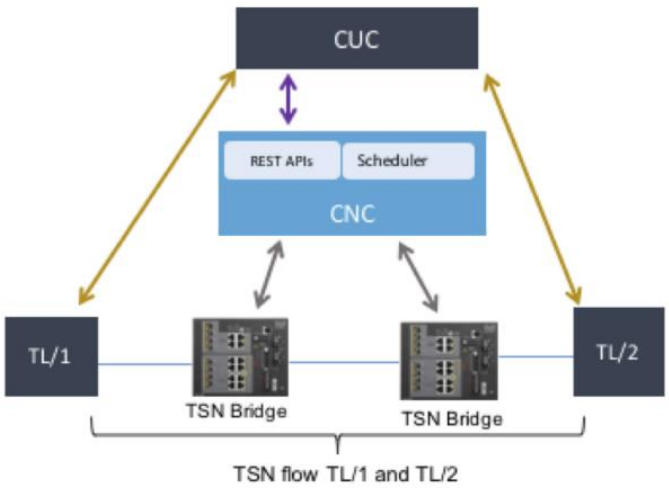


Figure 2.4 Cisco Proposal for CUC-CNC structure (taken from [13])

Using this simplified model as an example, the steps to create the TSN flow are described:

1. CUC Initiates Physical Topology Discovery: Using LLDP and a seed device, the CNC explores the physical topology, discovering each device and how they are connected. This includes the end devices that support LLDP.

2. CUC Requests Network Resources: Performed by the engineer responsible for defining the end to end communication. The engineer works out which end device (talker) has to communicate with other end devices (listener). The engineer is responsible for identifying all listeners, because there can be more than one for each TSN flow. The engineer can also define the latency requirements for the communication, the maximum size of the Ethernet packet that will be sent, and other dependencies (for example, whether there is a sequence order to the TSN flows). The CUC will gather this for all TSN flow requests and submit to the CNC using an API for accepting requests.

3. Compute Schedule: The engineer (via the CUC) will initiate a request that the CNC compute the schedule. The CNC will return success or failure for the request (depending on how the petitions can be or not fulfilled with the present network topology/elements). The end schedule cannot be computed unless the CNC knows the physical topology.

4. View Computation Results: Typically, the network engineer would want to view the schedule and verify before making it go live. The engineer (via the CUC) requests the CNC return the details of the computed schedule. This includes the details for each device involved in the TSN flows. The details include everything the end devices and bridges need to know for configuring TSN:
    - Unique identifiers for each TSN flow (destination MAC address, VLAN, CoS)
    - Start and end of transmit window at each hop (talkers and bridges)
    - Start and end of receive window at each hop (listeners and bridges)
    - End-to-end latency as computed

5. Distribute Schedule: Satisfied that the schedule will work, the engineer (via the CUC) or an automatic process issues a request to the CNC to distribute the computed schedule to the TSN bridges. The CUC will also program the talkers and listeners for the TSN flows. The talkers are expected to transmit every TSN flow according to a schedule.

2.2.2.2 National Instruments TSN Configuration Roadmap (July 2017)

NI presented in [14] a global view for TSN configuration, both centralized and distributed, in order to understand the main gaps that future TSN projects need to fill to create a working CNC.

The following section summarize the most important aspects of [14] explains about the future of TSN central configuration. A fully centralized scenario using HTTPS requires the definition of a complete set of YANG models for CNC. There are some gaps in the CNC YANG definition; we now list briefly the missing ones.

Hierarchy in CNC: each CUC can work on multiple domains (separated TSN networks with CUC's end stations), and each CNC can support multiple CUCs. There is the need to define a working hierarchy for networks with multiple CUC that use the same CNC.

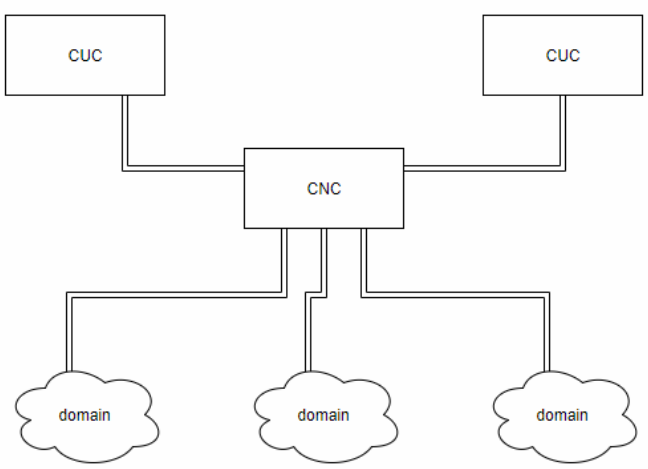


Figure 2.5 CNC hierarchy

CNC Actions:

- Topology discovery (LLDP exploration, discovering capabilities), there is some proposals of a LLDP YANG model at this moment [12].
- Computing reservations of scheduled traffic reservations, considering latency analysis and frame preemption.
- Configuring the computed reservations in bridges.

Topology and capabilities:
YANG model used both for topology display in CUC and offline topology design. A proposal is to improve and expand I2RS topology YANG model [15] with LLDP local info and TSN bridge capabilities.

Inter stream delay:
Minimum time between the transmission of two streams that ensures the correct transmission of a certain “F” time-aware transmission.

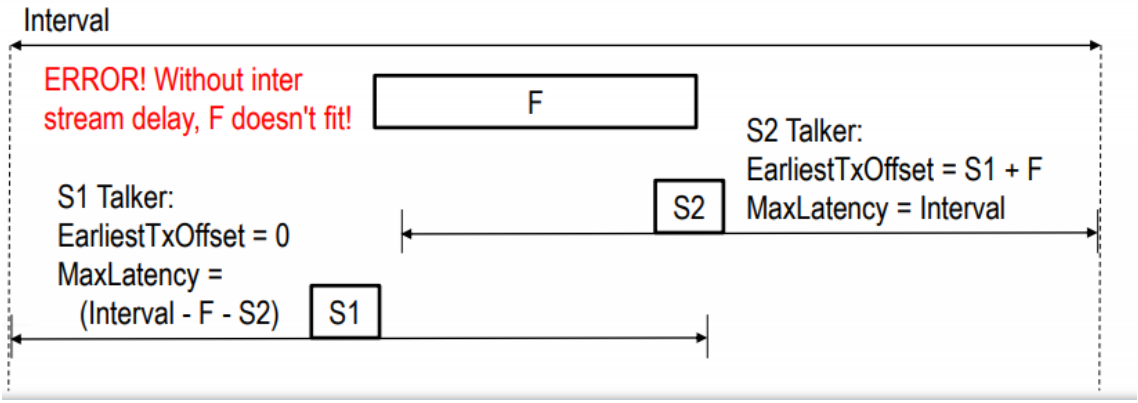


Figure 2.6 Inter Stream Delay (taken from [14])

Time synchronization:
YANG model needed to integrate information models related to time synchronization, that can be profile-independent (using any of the multiple IEEE 1588 profiles).

Confederation of multiple CNC:
YANG model needed for streams that cross CNC domains. It is proposed for a possible 802.1Qcc revision.

As of June 2018 some of these required YANG models have provisional versions, the most recent of them are described further in this document.

2.2.2.3. YANG models for CNC by TTTech (January 2017)

This proposal information for new YANG models can be found both in the standard 802.1ABcu (LLDP YANG Data Model, last updated December 2017) [41] and 802.1Qcw (YANG Data Models for Scheduled Traffic, Frame Preemption, and Per-Stream Filtering and Policing ,last updated December 2017) [42].

The YANG models needed for a CNC implementation are:

- Description of the API part of 802.1Qcc [23].
- Data format for the input of the CNC.
- Data format for the output of the CNC.

The status for these models as January 2017 classified by standard is:

- There are experimental data models for the 802.1QBv [28].
- The data models for 802.1CB [32], 802.1QCi [46] and 802.1AS-rev [44] do not exist.

The status for the models classified by the kind of information contained:

- Topology YANG models are based on LLDP MIBs. The experimental module by NI [12] is used by the SoC-e MTSN Kits.
- The datastream data models are covered by 802.1Qcc [23].
- Device parameters: YANG modules for 802.1Qcp[43] needs additional parameters as switch delay, that are defined in 802.1Qcc[23]. Other parameters are defined by the 802.1 standards QBv [28], As-rev[44], CB [32] and Qbu [45].
- Clock Synchronization data models will be bases in the module for 802.1 AS-rev[44].

2.2.3. Conclusions on the state of the elements needed for the CNC

As a summary for the 2.2.1 and 2.2.2 sections, the following list presents the state of the elements needed to develop a CNC. This will help us to create a proposal.

- The network topology discovery (including capabilities) method exists (LLDP).
- The YANG models to store the discovered topology and the hardware configuration, including TSN specific capabilities are being currently developed. Some prototypes are already created and can be tested.
- The protocol to communicate with the hardware is NETCONF or RESTCONF.
- The method to communicate with the TSN software can be a REST API using HTTP messages.
- The algorithms to compute the TSN scheduling are being developed. Some proposals like the No-wait packet scheduling are created but need further improvements (see Annex 6).
- The CUC for the user and applications to interact with the CNC can be created based on SDN controllers and web applications.

2.2.4. Proposal for the implementation of a CNC

Having identified all the necessary elements and having acquired practical experience with real TSN hardware implementations, the following elements can be chosen to create a CNC proposal.

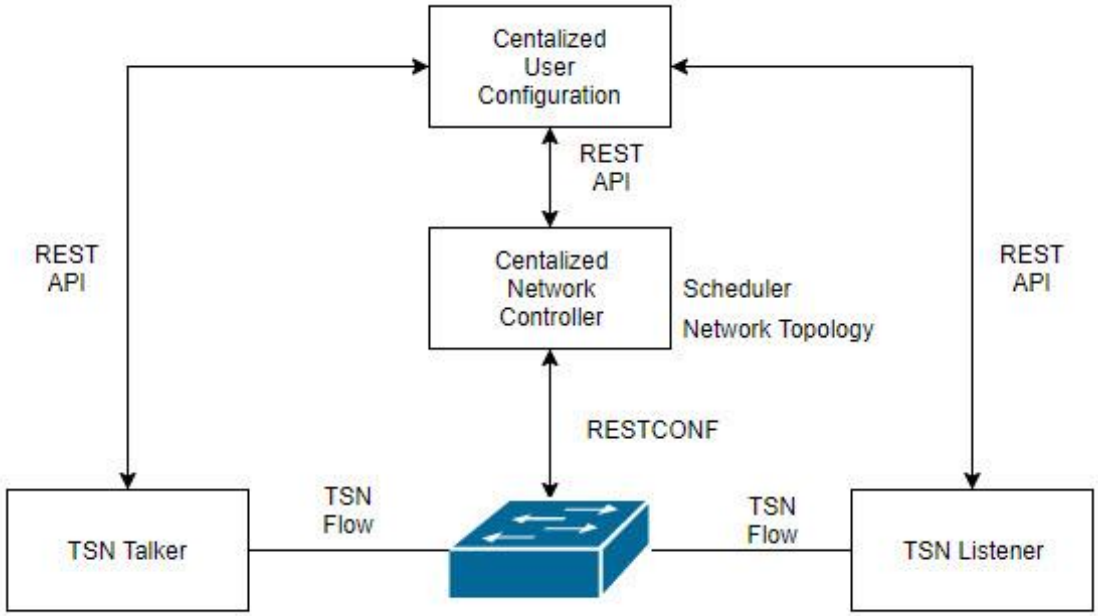


Figure 2.7 Proposed CNC-CUC architecture

First of all the existing elements that are working at the EETAC's TSN environment are listed. More details can be found in Chapter 3 and Appendix 1.

- RESTCONF: Using Jetconf, a RESTCONF distribution written in Python, [22], to work as a RESTCONF server, and the CURL tool [51] to build HTTPS messages to make the petitions to the Jetconf server.

- YANG models specific for TSN/CNC: ieee802-dot1q-sched [21], ieee802-dot1q-lldp [12].

- LLDP agent: lldpd [10], a LLDP free distribution that can be used as a Linux daemon.

Aside from the existing components, to fill the gaps needed to complete the CNC, the next recommended components are proposed:

- Scheduling Algorithm: No-wait Packed Shceduling for TSN exists on a conceptual level [17], but the integration with the CNC is needed.

- Connection with the CUC: REST API designed to share messages between the CUC and the CNC. In the lab hardware these messages are transmitted via Websocket messages sharing the configuration parameters via plain text. A more complete CNC version needs a HTTPS server/client to handle the petitions in a secure communication. The connection between the TSN-enabled applications and the CUC will need too an API designed to offer simple methods to allow this communication.

- Synchronization: Synchronization needs to be done in advance via PTP, and it is not considered as a CNC function in this work. The hardware hosting the CNC can be a PC, a virtual machine running on a PC, or any hardware that can run a light operating system like Debian Linux.

The following sections describe how to enhance and implement every aspect of the existing TSN Central configuration tools in order to create a CNC prototype, focused on the Time-Aware Shaper central configuration.

2.2.4.1. Topology discovery

Considering a scenario that uses similar hardware to the equipment described in Chapter 3, our TSN-enabled equipment (that can work as a switch or a router) runs a Linux distribution. This makes using the lldpd LLDP agent the best course of action. All the TSN machines need a lldpd agent that discovers their neighbors, and using the "*ieee802-dot1q-lldp"* YANG model format, the information gathered by LLDP can be sent to the controller using a RESTCONF HTTP message. That means that all the TSN enabled hardware on the network needs to have a Jetconf RESTCONF server installed in order to send and receive configuration messages.

One solution to gather all the topology information can be providing the controller with an option to do a neighbor discovery using LLDP, and then flood all the discovered neighbors with RESTCONF petitions to get their LLDP YANG models.

The TSN equipment needs to be enhanced by another YANG model that is not yet included in our lab's machines, the TSN capabilities model. In this model, the specific TSN capabilities of our hardware is described, this includes the maximum number of time slots supported by the hardware. The simplest way of implementing this description without waiting for a new model (or creating it) is adding a specific field in the "*ietf-interfaces"* YANG model [52], that is included in our lab hardware and all network equipment compatible with NETCONF or RESTCONF. This element can be a leaf associated to an specific network interface.

```
leaf timeslots {type string}
```

Together with the LLDP YANG model, the controller must ask about every machine's capabilities using RESTCONF messages. The list of machines and

capabilities must be parsed in order to create a logical and easy to understand file that contains all the topology information.
An example can be found in [18], describing nodes and links, but it needs to be enhanced with TSN information, in our case the maximum allowed time slots.

```
<topology>
    <nodes>
        <node id="router1.foo.net">
            <rid>10.0.0.1</rid>
            <interfaces>
                <interface id="10.0.2.0/30">
                    <ip mask="10.0.2.0/30">10.0.2.1</ip>
                </interface>
            </interfaces>
          <max time slots =32>
</node> ...
    </nodes>
    <links>
        <link id="1 -> 2">
            <from if="10.0.2.0/30" node="10.0.0.1"/>
            <to if="10.0.2.0/30" node="10.0.0.2"/>
        </link>
        ...
    </links>
</topology>
```

This information can be sent to the CUC in order to use a graphical tool to show the topology. SDN controllers like OpenDaylight are the way to go in order to deliver this useful information to the end user Figure 2.8 and Figure 2.9 show the two parallel processes needed for the network discovery.

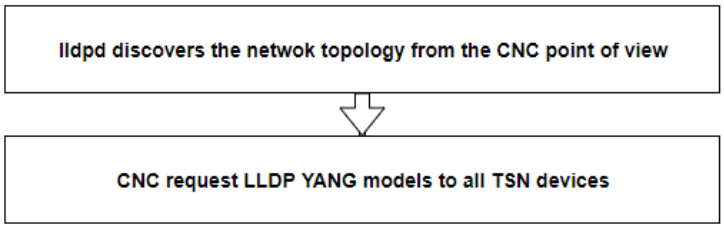


Figure 2.8 LLDP discovery from the point of view of CNC

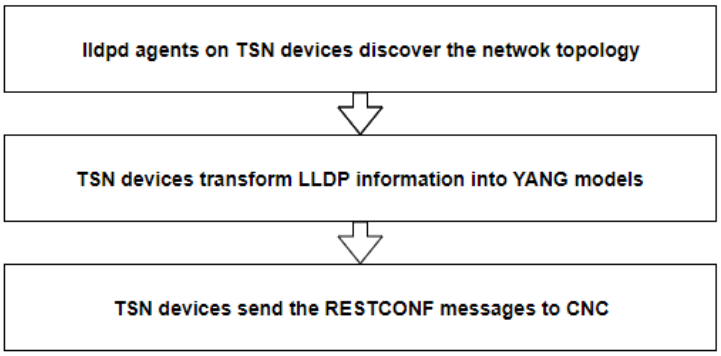


Figure 2.9 LLDP discovery from the point of view of TSN devices

2.2.4.2. CUC communication

End users can benefit from a graphical interface to configure the CNC operations when needed, even if the main goal of the CNC is to make the TSN configuration an automatic process. This interface and the connection between TSN applications and the CNC are the main pieces of a simplified CUC.

A REST API that defines the operations that the CNC allows, inspired on any web service, is the most efficient way to integrate software that sends TSN traffic with the CUC. In the EETAC laboratory, the web service interface sends Websockets messages to the hardware, delivering the configuration in plain text, and then sending RESTCONF messages using the parsed plain text information in a YANG model. The MTSN Kits web interface allows both the configuration of the time-aware and credit based shaper and the visualization of the network topology.

Another example is the Cisco prototypes for CNC, that have a web interface for both the graphical representation of the topology and the creation of TSN flows, as shown in Figure 2.11

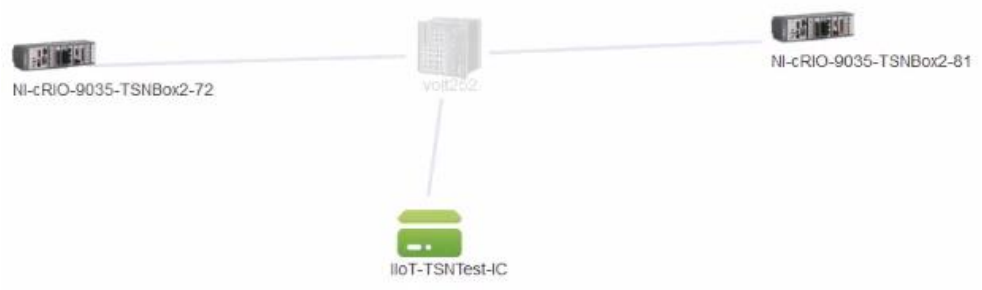


Figure 2.10 Cisco's CNC network topology

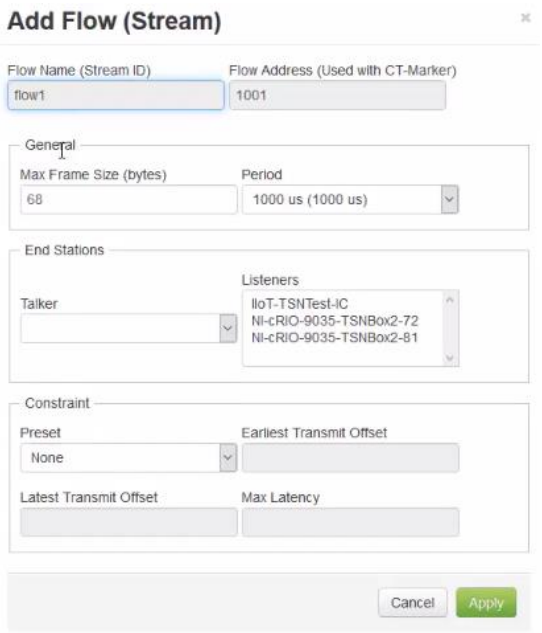


Figure 2.11 Cisco's CNC flow creation

Both examples set the tools to configure TSN capabilities using a web interface, but the real solution needed to create a real CNC is an API that can be integrated with TSN software and can communicate the flow needs and restrictions directly to the controller, without the need of human operation.

HTTPS is a good solution to communicate between CUC and CNC, with the only requirement as basic server in both CNC and CUC to handle the messages and listen for them, as shown in Figure 2.12.

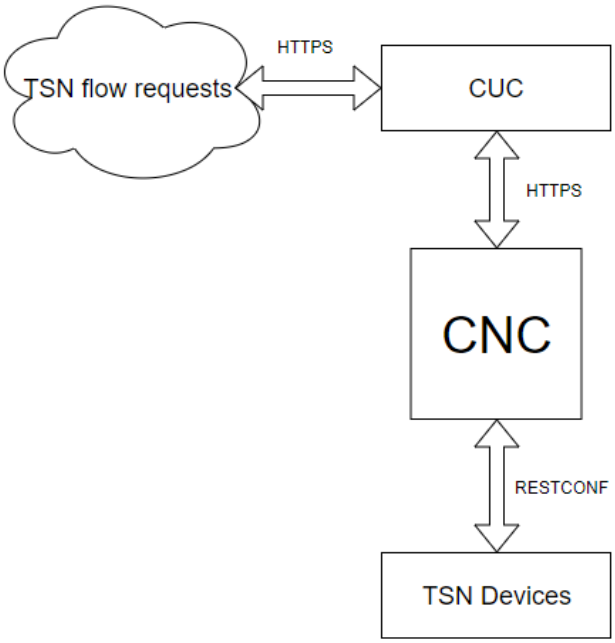


Figure 2.12 Communication protocols in a TSN Centralized network

2.2.4.3. Schedule computing and delivery

Using a real TSN scheduler based on a scheduling algorithm is the hardest and most important piece to create an independent CNC. When the CNC has both the network topology including TSN capabilities and the desired TSN flow restrictions describing the desired flows, it needs to parse this data in a format that can be used by a specific algorithm.

In the most basic scenario, the data the algorithm needs in order to create the schedule is, as illustrated in Figure 2.13.

- Maximum number of time slots supported by the hardware.
- Number of different traffic classes mapped to VLAN PCP priorities.
- Maximum available bandwidth.
- Bandwidth required by each TSN flow.

A network with only a few nodes could use a software implementation of the no-wait packet scheduling for TSN described in Annex 6 of this thesis. The same Annex also describes in more detail the network constraints and how this scheduler proposal escalates depending on the number of network nodes. Scalability is a problem that needs further experimentation considering that the future of TSN includes techniques like the frame pre-emption or path redundancy that will help building better and more robust TSN networks, but can increase the computational cost of calculating the time-aware schedule.

The last step will be sending RESTCONF messages including *ieee802-dot1q-sched* YANG models with the computed schedules, featuring time slot distribution and allowed VLAN traffic in each time slot. As all the TSN switches on a network must share the same schedule, this message can be flooded to all TSN enabled machines in the network.

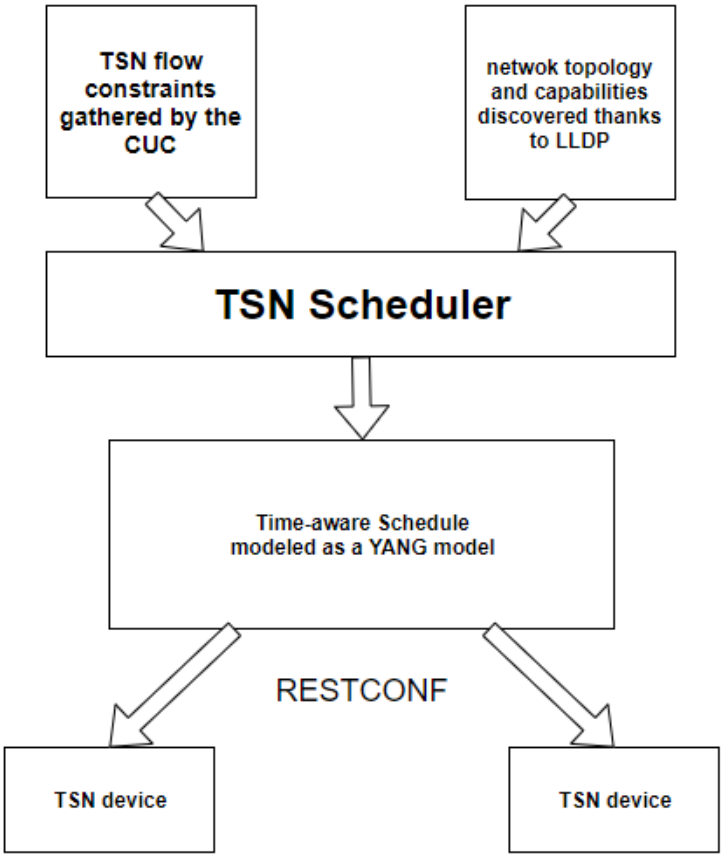


Figure 2.13 Main inputs and outputs from the TSN Scheduler

CHAPTER 3. TESTS OF RESTCONF AND LLDP WITH REAL TSN EQUIPMENT

In this chapter we will briefly describe the equipment used in the project and how this equipment implements some of the CNC elements explained in Chapter 2, RESTCONF and LLDP. For more information about the specifics of the hardware used in the project, the Annex 5 has a more detailed description and the Annexes 1, 2, 3 and 4 contain information in how to use, configure and modify the TSN capabilities of the equipment.

3.1 Overview of the MTSN Zynq and MPSoC Kits

For the purpose of a practical understanding of real implementations of some TSN standards, the MTSN (Multiport TSN) Kits equipment from SoC-e (System On Chip Engineering) [49] was tested in the controlled environment of UPC EETAC's networking laboratory. A section about the usage and demo capabilities of the hardware can be found in this document in Annex 1. It is advised to read Annex 5 in this document before the next sections, as this annex explains both the characteristic of the MTSN Kits and the way it handles the configuration of TSN capabilities using a web interface.

3.2. Use of RESTCONF

In order to change the configuration of the hardware, including the TSN capabilities, the MTSN Kits have installed the Jetconf RESTCONF distribution. Jetconf presents a pure Python implementation of a RESTCONF server.

The following YANG modules are featured by our version of MTSN Kit's Jetconf server, with the possibility of expanding this list in the future.


```
{
  "ietf-yang-library:modules-state": {
    "module-set-id": "9a9b7d2d28d4d78fa42e12348346990e3fb1c1b9",
    "module": [
      {
        "name": "ieee802-dot1q-sched",
        "namespace": "http://example.com/ieee802-dot1q-sched",
        "feature": [ "scheduled-traffic" ],
        "revision": "",
        "conformance-type": "implement"
      },
      {
        "name": "ieee802-dot1q-lldp",
        "namespace": "http://example.com/ieee802-dot1q-lldp",
        "feature": [ "lldp-tx","lldp-rx" ],
        "revision": "",
        "conformance-type": "implement"
      },
      {
```

```
        "name": "ietf-yang-types",
        "namespace": "http://example.com/ietf-yang-types",
        "revision": "2013-07-15",
        "conformance-type": "implement"
      },
      {
        "name": "ieee802-dot1q-types",
        "namespace": "http://example.com/ieee802-dot1q-types",
        "revision": "",
        "conformance-type": "implement"
      },
      {
        "name": "ietf-interfaces",
        "namespace": "http://example.com/ietf-interfaces",
        "revision": "",
        "conformance-type": "implement"
      },
      {
        "name": "ieee802-dot1q-bridge",
        "namespace": "http://example.com/ieee802-dot1q-bridge",
        "revision": "",
        "conformance-type": "implement"
      },
      {
        "name": "ieee802-types",
        "namespace": "http://example.com/ieee802-types",
        "revision": "",
        "conformance-type": "implement"
      },
      {
        "name": "iana-if-type",
        "namespace": "http://example.com/iana-if-type",
        "revision": "",
        "conformance-type": "implement"
      }
    ]
  }
}
```


Aside from the IEEE 802 modules needed by most network hardware, this implementation has the "*ieee802-dot1q-sched*" [12] module, that describes how to store the data to configure the Time-Aware Shaper on the MTSN Kit ports.

Each MTSN Kit stores the information regarding its own Time-Aware Shaper configuration and the LLDP remote systems in the same network in a file named "*data-qbv.json"* that is located in the Jetconf folder, formatted as a JSON file to make easy to share this information using RESTCONF.

Using the CURL tool [51], HTTPS requests can be sent to obtain and change the scheduler's configuration. In this case, the information will be encoded using the JSON format. The next example shows how to use the CURL tool to configure the time-aware scheduler. Other tools, like the well-known Postman software, can be used to generate the HTTP petitions, but the HTTPS restriction of using a client certificate makes CURL the easiest way of generating the messages while specifying the use of client cert.

The URL string follows the RESTCONF structure to access a specific datastore, and the CLIENT_CERT specifies where the authentication cert is located to enable the HTTPS connection. The first URL used specifies the “jetconf:conf-start” operation, needed to start the re-configuration process.

```
#!/bin/bash

PROXY="--noproxy 192.168.4.65"

CLIENT_CERT="./client_ssl/client_cert_curl.pem"

POST_DATA='{ "jetconf:input": {"name": "Edit 1"} }'
URL="https://192.168.4.65:8443/restconf/operations/jetconf:conf-start"
curl $PROXY -v --http2 -k --cert-type PEM -E $CLIENT_CERT -X POST -d
"$POST_DATA" "$URL"

POST_DATA=' {
        "interface": [
        {
          "name": "0",
          "type": "EthernetCsmacd",
          "enabled": true,
          "ieee802-dot1q-bridge:bridge-port":
          {
                  "ieee802-dot1q-sched:gate-parameters":
                  {
                          "admin-base-time":
                          {
                                  "fractional-seconds": 0,
                                  "seconds": 0
                          },
                          "admin-cycle-time":
                          {
                                  "denominator": 500000,
                                  "numerator": 8388
                          },
                          "config-change": true,
                          "admin-control-list-length": 3,
                          "admin-control-list": [
                          {
            "index": 0,
            "sgs-params":
                          {
            "gate-states-value": 1,
            "time-interval-value": 4194304
                          },
                          {
            "index": 1,
            "sgs-params":
                                  {
            "gate-states-value": 12,
            "time-interval-value": 4194304
                                }},
                          {
           "index": 2,
           "sgs-params":
                                  {
           "gate-states-value": 0,
           "time-interval-value": 4194304
```

```
                                } },
                            {
            "index": 3,
            "sgs-params":
                                {
            "gate-states-value": 2,
            "time-interval-value": 4194304
                                }}]}}},


URL="https://192.168.4.65:8443/restconf/data/ietf-
interfaces:interfaces"
#curl $PROXY -v --http2 -k --cert-type PEM -E $CLIENT_CERT -X POST -d
"$POST_DATA" "$URL"
curl $PROXY -v --http2 -k --cert-type PEM -E $CLIENT_CERT -X PUT -d
"$POST_DATA" "$URL"
URL="https://192.168.4.65:8443/restconf/operations/jetconf:conf-
commit"
curl $PROXY -v --http2 -k --cert-type PEM -E $CLIENT_CERT -X POST
"$URL"
```

The interesting part about the example, is the POST_DATA block, where the data sent by the POST request is written, always following same format as the "*ieee802-dot1q-sched*" YANG model.

The example shows how to configure one port. To do the same for the other hardware ports, repeat all the block inside the "*interface*" section, changing the "name": "0" to "name":"X" where X is the port number. The cycle time can be calculated using the numerator and denominator in the cycle time field. The obtained result is 0,016776. Dividing this number by 4 (because there's 4 time slots during each cycle). We obtain 0,004194 seconds, the same value can be found at each time slot time interval field: 4194304 nanoseconds, confirming our previous calculation.

The timeslots (4 in our hardware implementation) are numbered from 0 to 3, this value can be found in the "index" lines. The last interesting field is "*gate-states-value*", an uint8 (unsigned integer with 8 bits) that represents the 8 priority traffic classes on each time slot.

The last URL specifies the "*jetconf:conf-commit*" operation, needed to write the configuration in the datastore and then re-configure the hardware.

In order to view the Time-Aware Shaper configuration, the next HTTP CURL petition is used:

```
URL="https://192.168.4.65:8443/restconf/operations/jetconf:conf-start"
curl $PROXY -v --http2 -k --cert-type PEM -E $CLIENT_CERT -d
```

The response includes the Time-Aware Shaper configuration data written in the same format as the POST example.

In order to enhance the Jetconf server supported modules, the new YANG data model to be used must be added into the *"yang-data"* folder, also there's the

need to modify the *"yang-library-data-qbv.json"* file to enable the use of the new data model. This file contains the list of supported modules showed at the start of this section.

When using the web interface to configure the time-aware scheduler, Websocket messages send the desired configuration from the user PC to the MTSN Kits, using plain text via TCP port 3333. The next WebSocket message is transmitted between the laboratory PC and the MTSN Kit directly connected to it, with the objective of configuring another MTSN Kit that is connected to the TSN network.

```
192.168.4.99   192.168.4.64 WebSocket      445 WebSocket Text [FIN] [MASKED]
```

```
42["congigureDevice",[{"ipAddress":"192.168.4.65",
"ports":[0,1,2,3],
"refTime":"0",
"windowLength":"12000",
"numTimeSlots":"3","timeSlots":[{"index":0,"sgs-
params":{"gate-states-value":7,
"time-interval-value":4000}},{"index":1,"sgs-
params":{"gate-states-value":56,
"time-interval-value":4000}},{"index":2,"sgs-
params":{"gate-states-value":192,
"time-interval-value":4000}}]}]]
```

It has information about the Time-Aware Shaper configuration, that is received by the configuration servers in the MTSN Kit, that creates the HTTP request to perform a RESTCONF operation.

The MTSN Kits have a Javascript function listening to the 3333 port, that is designed to select which operation to perform depending on the first string sent via Websockets.

```
socket.on('congigureDevice', function(data){
for(i = 0; i< data.length; i++){
var ip_address = data[i].ipAddress;
var bridge_ports = data[i].ports;
var bridge_port_data=
JSON.stringify(parse_interfaces_data(data[i]));

restconfClient.updateSchema(bridge_ports.length,bridge_port_data,
bridge_ports,ip_address,http2Connection)}
```

In this example, the operation "congigureDevice" (there is a typo in this function name, but we maintain the original name in this document to prevent confusion), that is the operation designed to trigger Time-Aware Shaper changes in this version, parses the desired elements and follows with the operation "updateSchema". "updateSchema" is a function included in a Javascript class created to work as a RESTCONF client, and follows a chain of functions destined to start the configuration, create the request and send it and commit the changes. The request includes a PUT message to re-configure each network interface. This functions define the HTTP operation, the desired URL and the use of parsed data.

Note that this Javascript class is heavily focused on the Time-Aware Shaper configuration, to expand the capabilities of the RESTCONF client this class needs to be enhanced with any new operation.

The MTSN Kits store a log with all the RESTCONF messages. Annex 4 includes a list of captures messages that show both the retrieval of datastore information and the change of TSN parameters configuration. This log is configured to store all RESTCONF HTTPS packets, and is especially useful because the HTTPS messages are encrypted, making harder to observe their content via Wireshark or tcpdump captures.

3.3. Use of LLDP

LLDP is the protocol used by the MTSN Kits in order to discovery the network topology. Instead of using an LLDP MIB, the MTSN Kits evaluate the LLDP information and store it using the specified YANG model described in [12]. Requesting a MTSN Kit about its “remote systems” shows the information this machine has about the surrounding LLDP enabled machines. The first version of the MTSN Kits does not support LLDP, so connecting non-upgraded machines to the network will not show the machine in the LLDP stored topology

The next messages show a request for network topology. In this scenario, the machine with IP address 192.168.4.65 returns a system list with information about the machine with IP address 192.168.4.64, the only other compatible hardware in the tested network.

```
---------------------------------REQUEST-----------------------
GET https://192.168.4.65:8443/restconf/data/ieee802-dot1q-lldp:remote-
systems

-----------------Data------------------

--RESPONSE----------------------
---------------Headers-----------------
:status: 200
content-type: application/yang.api+json
content-length: 696
server: jetconf-h2
cache-control: No-Cache
access-control-allow-origin: all
access-control-allow-headers: Content-Type
etag: 1745839175
last-modified: Sat, 21 May 2016 22:37:16 GMT
---------------------------------------

-----------------Data------------------
{
    "ieee802-dot1q-lldp:remote-systems": {
        "system-list": [
            {
                "time-mark": 288,
                "port-desc": "b'eth1.1'",
                "port-id": "NzA6Zjg6ZTc6ZDA6MTo2ZQ==",
```

```
                "index": 1,
                "interface": "1",
                "sys-name": "b'SoC-e'",
                "sys-desc": "b'MTSN Switch'",
                "man-addr-list": [
                    {
                        "rem-man-addr": "MTkyLjE2OC40LjY0",
                        "rem-man-addr-subtype": 1
                    }
                ],
                "dest-address-index": 1,
                "chassis-id": "MDphOjM1OjA6MToyMg=="
            }
        ]
    }
}
```


The web utility provided by SoC-e allows the graphical representation of the network topology, created using some basic Javascript functions, as illustrated in Figure 3.1.

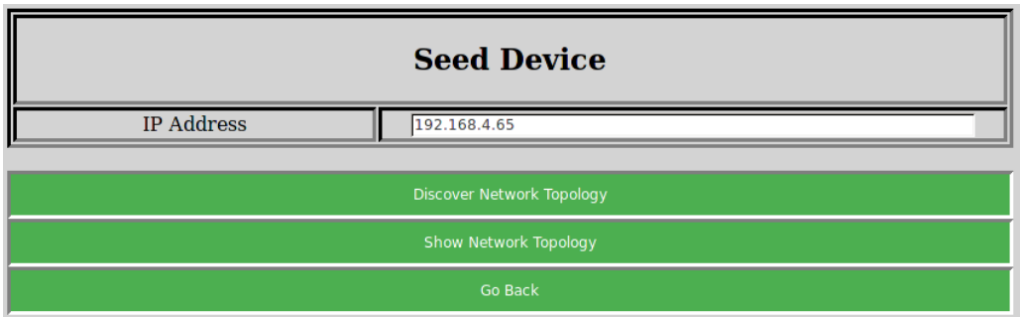


Figure 3.1 Topology discovery options for the SoC-e web-based interface

shows the view that the MTSN Kit with IP address 192.168.4.64 has, directly connected to the other compatible MTSN Kit and showing it's own loopback interface. The seed device chosen before the network topology discovery process is the device that will start the spread of LLDP discovery messages.

Figure 3.2 Graphical representation of the network topology

As the information captured in the RESTCONF petition is not in a format easy to understand, a more in depth study of lldpd [20], the LLDP Linux daemon used to capture the topology information in this scenario, can shed some light into the matter.

The lldpd service has a configuration file LLDPD.conf, where the service is detailed. The most interesting fields from this file are both the interval between LLDP messages, that is 30 seconds in our scenario, and the "unconfigure lldp

capabilities-advertisements" that prevents the configuration files from overriding the original system capabilities advertisements.

```
configure system hostname "SoC-e"
configure system description "MTSN Switch"
configure system interface pattern eth1.0,eth1.1
configure system ip management pattern 192.168.4.64
configure lldp tx-interval 30
unconfigure lldp capabilities-advertisements
```

lldpd allows the user to configure the LLDP protocol using a CLI, that is booted using the *lldpcli* command in the Linux terminal. This commands fall into two main categories: show and configure. The following list of commands and responses is aimed to the understanding on how lldpd works and how the laboratory equipment is configured.

show neighbors

```
LLDP neighbors:
-------------------------------------------------------------------
Interface:    eth1.1, via: LLDP, RID: 1, Time: 0 day, 00:27:05
  Chassis:
    ChassisID:    mac 00:0a:35:00:01:22
    SysName:      SoC-e
    SysDescr:     MTSN Switch
    TTL:          120
    MgmtIP:       192.168.4.64
  Port:
    PortID:       mac 70:f8:e7:d0:01:6e
    PortDescr:    eth1.1
```

This list describes the discovered neighbors and the discovered ports from this neighbors. It only lasts for 120 seconds as defined in the TTL field, so the information is constantly being updated.

show configuration

```
Global configuration:
-------------------------------------------------
Configuration:
  Transmit delay: 30
  Transmit hold: 4
  Receive mode: no
  Pattern for management addresses: 192.168.4.64
  Interface pattern: eth1.0,eth1.1
  Interface pattern for chassis ID: (none)
  Override description with: MTSN Switch
  Override platform with: Linux
  Override system name with: SoC-e
  Advertise version: yes
  Update interface descriptions: no
  Promiscuous mode on managed interfaces: no
  Disable LLDP-MED inventory: yes
  LLDP-MED fast start mechanism: yes
```

```
  LLDP-MED fast start interval: 1
  Source MAC for LLDP frames on bond slaves: local
  Port ID TLV subtype for LLDP frames: unknown
  Agent type:   unknown
```

This list shows the parameters that determine how the LLDP protocol is used, overriding from the default configuration fields when necessary.

show chassis

```
Local chassis:
-------------------------------------------------------------------
Chassis:
  ChassisID:    mac d8:80:39:67:19:80
  SysName:      SoC-e
  SysDescr:     MTSN Switch
  TTL:          120
  MgmtIP:       192.168.4.64
  Capability:   Bridge, off
  Capability:   Router, off
  Capability:   Wlan, off
  Capability:   Station, on
```

The chassis list shows the description of both the interfaces and capabilities of the local machine. See how the bridge, router and wlan options are turned off.

The Jetconf server has a class dedicated to obtain the LLDP information to store it in a YANG model, but without reformatting the data. As the next Python code shows, the server obtain the data from the lldp_data Linux directory and stores it with the prefix *ieee802-dot1q-lldp:remote-systems*, that is a YANG model prefix, and adding *system-list* before the obtained list.

```
import json

EXTERNAL_PORTS = 2

def get_lldp_data():
    data_ifaces = []
    for x in range(0, EXTERNAL_PORTS):
       try:
       if_data= json.load(open('/opt/sysdoc/lldp_data/eth1.'+str(x)))
       data_ifaces.append(if_data)
       except OSError as e:
       pass
       data = { "ieee802-dot1q-lldp:remote-systems" : {
       "system-list" : data_ifaces
}
}
    return data
```

Investigating the file that the server parses, the network information is coded exactly like the lists that RESTCONF distributes, at the end the server creates the RESTCONF petitions using the same file.

```
"index": 1,
    "sys-desc": "b'MTSN Switch'",
    "man-addr-list": [
        {
            "rem-man-addr-subtype": 1,
            "rem-man-addr": "MTkyLjE2OC40LjY0"
        }
    ],
    "port-desc": "b'eth1.1'",
    "dest-address-index": 1,
    "time-mark": 288,
    "interface": "1",
    "port-id": "NzA6Zjg6ZTc6ZDA6MTo2ZQ==",
    "chassis-id": "MDphOjM1OjA6MToyMg==",
    "sys-name": "b'SoC-e'"}root@/opt/sysdoc/lldp_data#
```


SoC-e's implementation uses TLV (type-length-value) encoding [56], and parses the LLDP messages using this method in their sniffing LLDP method.

```
address_string_length=int.from_bytes(tlv_datafield[0:1],
byteorder='big')
"address_string_length" : address_string_length,
```

CONCLUSIONS

Conclusions

The actual trend of centralized controlled networks, inspired by SDN, and the need of a universal solution and standards for deterministic Ethernet networks presents us with a big opportunity: unite the two technologies to create a centralized deterministic Ethernet network.

The biggest networking hardware manufacturers are already on the work (Cisco, TTTech, National Instruments…) and the IEEE has created a specific group within the Time-Sensitive Networking Task Group, working in the IEEE 802.1Qcc standard in order to improve stream reservation in TSN networks.

The elements needed for developing the technology are already there: network discovery tools, scheduling algorithm proposals, and this thesis tries to specify how to use these technologies according to the needs of TSN networks.

Our proposal includes both the most interesting protocol implementations and software elements to create a centralized controller and the most convenient way of creating the communication between the TSN hardware and the centralized controller. We have accomplished the deployment of a small TSN network based on a mix of open and vendor-specific set of tools installed on a Lubuntu OS, that allowed us to test and understand a very basic CNC approach based on a Webservice and the RESTCONF configuration of the hardware.

The strong aspects of this investigation are both an overall and clear view of which steps must be followed in order to implement a CNC, a defined set of existing protocols and implementations and the practical experience using real TSN equipment that can allow for an easy and fast introduction to following developments. The creation of a CNC will benefit all Ethernet-based networks: IT, IoT and will be helpful to help the integration of OT networks in Ethernet environments, as described in Annex 7.

On the other hand, there are a few limitations. Hardware that is controlled using an open operating system like Debian is a must to use and implement open-standard and protocol implementations like lldpd or Jetconf, because of that limitation integrating TSN equipment that is not open or uses vendor specific protocols or implementations will heavily depend on the capabilities of the equipment. Aside from that, the real implementation of the project proposal has a big programming focus. There is the need of programing a fair amount of functions to both parse data, handle HTTP messages and use algorithm, so a big part of the project is not TSN-focused, making the further development of the CNC a collaborative project between programming and knowledge of TSN protocols.

We can conclude that a centralized controlled TSN network can be a reality during the next months, not only because the main TSN hardware/software

vendors are already working on the matter, but also because as we studied the most realistic proposals and the software necessary to fulfill the requirements, we reached the conclusion that the CNC is a project heavily based in already existing technologies and network control paradigms like SDN.

This proposal has been already presented at some national events like the Salón BIT Audiovisual I+D [54] in the form of a poster and a paper has been accepted as one of the featured articles in the XXXIII Simposium Nacional de la Unión Científica Internacional de Radio [53].

Future lines of research

The following steps in this project are clearly about the real implementation and use of the existing elements and proposals in order to create a working prototype for a TSN CNC. The standardization of the CNC is a key piece not only in the IT networks, but also a key piece to advance the Industry 4.0 projects, the audiovisual communications and the spread of Ethernet networks on time sensitive communications as a whole.

To achieve this goal, there is three main lines of work. The first one is the collaboration with other TSN investigation teams. During this project, SoC-e's assistance both in the practical and theoretical aspects has been important to advance in the understanding of the TSN technology and how to implement this technology in a real scenario. Not only Soc-e, Cisco, TTTech, National Instruments or other networking hardware/software vendors are worth keeping an eye on. And the IEEE 802.1Qcc Task Group collaborates with all this vendors and other research teams, being the best source of information on the topic. As a more specific note, we would advise keeping contact with SoC-e's developing team. They are currently implementing new TSN capabilities, like the 802.1Qci [46], and are available for future cooperation.

The second line of work is a programming effort based on implementing a way to communicate the TSN hardware running Lubuntu with another machine, a PC or a Virtual Machine in order to share TSN constraints and topology information. In the proposal presented in Section 2.2.4. from this thesis, we concluded that we will need to develop the next components:

- Implement HTTPS handlers in both the TSN hardware and the controller.
- Define HTTPS messages that replace the Websocket messages that are currently being used to share information between TSN hardware and the CNC.
- Write Javascript functions to parse the desired information and transform it into RESTCONF petitions that will be sent to the TSN hardware.
- Find or create a TSN scheduler algorithm and implement it whit the other CNC components.

- Create a simple Web to allow the user to define the TSN constraints that the TSN scheduler will use to calculate the schedule.

- The enhancement of the Interfaces management YANG model [52] to share basic TSN capabilities, mostly the allowed time slots used in the Time-Aware Shaper (TAS).

These components, together with the already working ones, like the LLDP discovery and data parsing, the Jetconf RESTCONF server already working in the laboratory equipment and already implemented YANG models required for the TAS operation would serve to create the most basic CNC, that will serve only to auto generate schedules for the TAS depending on the user desired constraints and the TSN network topology and capabilities.

The third and last line of work will be the creation of a real CNC-CUC implementation for TSN. Using the previously described software tools, to expand them to develop the CNC-CUC scheme described in the CNC proposal there is the need to create a integration API to allow TSN compatible software to communicate with the CNC. For this purpose, the CUC, an expanded version of the web developed mostly for user interaction before this point, must share the integration API's endpoints, to being able to receive the information we will need from TSN compatible software: the TSN flow constraints that the software needs in order transmit the TSN flow with its desired timing requisites.

Environmental aspects

Considering that the main goal of this project is to define a future line of work to create a CNC implementation, and that line of work is based mainly on the creation or integration of software tools, I think that the environmental impact of the project is not relevant. The SDN trend allows us to create simpler devices thanks to the centralized control plane, reducing CAPEX and OPEX, and thus making the implementation of new networks cheaper and more echo-friendly, especially thanks to the re-use of hardware that centralized control allows. Another benefit of using centralized controlled networks is that the nodes that are not operating in a certain configuration can be shut down, saving energy in the process. The conclusion is that even if this thesis does not present any project that will have any major environmental aspect, being a part of the centralized control trend makes us consider that the end result will be fairly echo-friendly.

BIBLIOGRAPHY

[1] J. Lei Du and M. Herlic, “Software-defined Networking for Real-time Ethernet”, 2016 [Online] Available: https://www.salzburgresearch.at/wp-content/uploads/2016/05/Software-defined-Networking-for-Real-time-Ethernet.pdf

[2] Wikipedia, “OPC Unified Architecture”, last edited 2018 [Online] Available: https://en.wikipedia.org/wiki/OPC_Unified_Architecture

[3] BR Automation, “OPC UA TSN - Testeados y probados sobre el terreno”, 2017 [Online] Available: https://www.br-automation.com/es/sobre-nosotros/newsletter/actual/opc-ua-tsn-testeados-y-probados-sobre-el-terreno

[4] TTTech, “Time-Sensitive Networking (TSN) Products”, 2018 [Online] Available:https://www.tttech.com/technologies/deterministic-Ethernet/time-sensitive-networking/

[5] F. Castaño, O. Castaño, G. Anangono, D. Rincón, A. Agustí, “SDN-based Management of Time-Sensitive Networks in 5G Fronthaul”, paper accepted for publication in URSI 2018, XXXIII Simposium Nacional de la Unión Científica Internacional de Radio (URSI), Granada, 2018

[6] P. Didier and G. A. Ditzel III, “Time Sensitive Networks (TSN) technology overview”, 2017 [Online] Available: http://www.iebmedia.com/index.php?id=11824&parentid=74&themeid=255&showdetail=true&bb=true

[7] M. Bjorklund, “YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)” RFC 6020, 2010, [Online] Available: https://tools.ietf.org/html/rfc6020

[8] A. Bierman, M. Bjorklund, K. Watsen “RESTCONF Protocol” Proposed Standard, RFC 8040, 2017 [Online] Available: https://tools.ietf.org/html/rfc8040

[9] I. Romero, “CDP and LLDP”, Blog Irontec 2016 [Online] Available: https://blog.irontec.com/cdp-y-lldp-nuestros-grandes-aliados/

[10] V. Bernat, “Implementation of IEEE 802.1ab (LLDP)”, last version from 2018 [Online] Available: https://github.com/vincentbernat/lldpd/

[11] J. Schoenwaelder “Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules”, RFC 6643, 2012, [Online] Available: https://tools.ietf.org/html/rfc6643

[12] R. Cummings, “Add experimental module for IEEE Std 802.1AB2016 (LLDP)” ,last version from 2017 [Online] Available: https://github.com/YangModels/yang/blob/master/experimental/ieee/802.1/ieee802-dot1q-lldp.yang

[13] Cisco, “Time-Sensitive Networking:A Technical Introduction”, 2017 [Online]Available:https://www.cisco.com/c/dam/en/us/solutions/collateral/industry-solutions/white-paper-c11-738950.pdf

[14] R. Cummings, “TSN Configuration Roadmap by NI”, 2017 [Online] Available: http://www.ieee802.org/1/files/public/docs2017/new-cummings-tsn-config-roadmap-0717-v0.pdf

[15] A. Clemm, J. Medved, R. Varga, N. Bahadur, H. Ananthakrishnan, X. Liu, “A YANG Data Model for Network Topologies”, RFC 8345, 2018 [Online] Available: https://datatracker.ietf.org/doc/rfc8345/

[16] S. Craciunas and R. Serna, “An Overview of Scheduling Mechanisms forTime-sensitive Networks”, [Online] Available: http://www.cs.uni-salzburg.at/~scraciunas/pdf/techreports/craciunas_ETR17.pdf

[17] F. Dürr and N. Ganesh, “No-wait Packet Scheduling for IEEE Time-sensitive Networks (TSN)”, [Online] Available: ftp://ftp.informatik.uni-stuttgart.de/pub/library/ncstrl.ustuttgart_fi/INPROC-2016-32/INPROC-2016-32.pdf

[18] S. Balon “A standard XML format for a network topology representation”, 2008 [Online] Available: http://totem.run.montefiore.ulg.ac.be/doc/UserGuide/node4.html

[19] SoC-e “MTSN Kit User Guide”, 2018

[20] V. Bernat, “lldpd Usage”, 2018 [Online] Available: https://vincentbernat.github.io/lldpd/usage.html

[21] R. Cummings, “ieee802-dot1q-sched.yang”, 2017 [Online] Available: https://github.com/YangModels/yang/blob/master/experimental/ieee/802.1/ieee802-dot1q-sched.yang

[22] Sydesk, “Jetconf”, last version from 2017[Online] Available: https://github.com/syedsk/jetconf/

[23] P802.1Qcc.Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, [Online] Available: https://1.ieee802.org/tsn/802-1qcc/

[24] Audio Video Bridging (AVB) Task Group main page, [Online] Available: http://www.ieee802.org/1/pages/avbridges.html

[25] 802.1AS - Timing and Synchronization main page, [Online] Available: http://www.ieee802.org/1/pages/802.1as.html

[26] Time-Sensitive Networking (TSN) Task Group main page, [Online] Available: http://www.ieee802.org/1/pages/tsn.html and https://1.ieee802.org/tsn/

[27] Precision Time Protocol Software Configuration Guide by Cisco, [Online] Available: https://www.cisco.com/c/en/us/td/docs/switches/connectedgrid/cg-switch-sw-master/software/configuration/guide/ptp/b_ptp_ie2ku.html

[28] 802.1Qbv - Enhancements for Scheduled Traffic,
[Online] Available: http://www.ieee802.org/1/pages/802.1bv.html

[29] Time sensitive networking from Newelectronics, [Online] Available: http://www.newelectronics.co.uk/electronics-technology/time-sensitive-networking-is-set-to-meet-the-precise-communication-needs-of-the-industrial-iot-and-smart-cars/116514/

[30] P802.1Qcc – Stream Reservation Protocol (SRP) Enhancements and Performance Improvements, [Online] Available: https://1.ieee802.org/tsn/802-1qcc/

[31]O. Castaño, “Building and testing a Time Sensitive Networking environment”, Bachelor degree thesis, EETAC, UPC, July 2018

[32] P802.1CB – Frame Replication and Elimination for Reliability, [Online] Available: https://1.ieee802.org/tsn/802-1cb/

[33] Using Reverse PTP to distribute time in Software Defined Networks, [Online] Available: https://ieeexplore.ieee.org/document/6948702/

[34] Common Public Radio Interface Wikipedia article, [Online] Available: https://en.wikipedia.org/wiki/Common_Public_Radio_Interface

[35] R. Suárez, S. Sallent, D. Rincón, A. Agustí, “Fully Synchronous SDN (FSS): Towards Synchronization-as-a-Service in 5G Networks”, paper accepted on URSI 2018, XXXIII Simposium Nacional de la Unión Científica Internacional de Radio (URSI), Granada, 2018

[36] Synchronous Ethernet Wikipedia article, [Online] Available: https://en.wikipedia.org/wiki/Synchronous_Ethernet

[37] “OPC UA TSN: The future of communication has arrived”, [Online] Available: https://www.br-automation.com/cs/produkty/innovations-2018/opc-ua-tsn-iiot-solutions/opc-ua-tsn-the-future-of-communication-has-arrived/

[38] R. Enns, M. Bjorklund, J. Schoenwaelder, A. Bierman, “Network Configuration Protocol (NETCONF)”, [Online] Available:

https://tools.ietf.org/html/rfc6241

[39] Remote Procedure Call Wikipedia Article, [Online] Available: https://en.wikipedia.org/wiki/Remote_procedure_call

[40] Management information base Wikipedia Article, [Online] Available: https://en.wikipedia.org/wiki/Management_information_base

[41] S. Mansfield, “P802.1ABcu – LLDP YANG Data Model”, [Online] Available: https://1.ieee802.org/tsn/802-1abcu/

[42] M. Gutierrez, “P802.1Qcw – YANG Data Models for Scheduled Traffic, Frame Preemption, and Per-Stream Filtering and Policing”, [Online] Available: https://1.ieee802.org/tsn/802-1qcw/

[43] M. Holness, “P802.1Qcp – Bridges and Bridged Networks Amendment: YANG Data Mode”, [Online] Available: https://1.ieee802.org/tsn/802-1qcp/

[44] G. Garner, “P802.1AS-Rev – Timing and Synchronization for Time-Sensitive Applications”, [Online] Available: https://1.ieee802.org/tsn/802-1asrev/

[45]”802.1Qbu - Frame Preemption”, [Online] Available http://www.ieee802.org/1/pages/802.1bu.html

[46] T. Jeffree, ” P802.1Qci – Per-Stream Filtering and Policing”, [Online] Available: https://1.ieee802.org/tsn/802-1qci/

[47] Satisfiability Modulo Theories Wikipedia Article, [Online] Available: https://en.wikipedia.org/wiki/Satisfiability_modulo_theories

[48] Tabu Search Wikipedia Article, [Online] Available: https://en.wikipedia.org/wiki/Tabu_search

[49] MTSN Kit main page, [Online] Available: https://soc-e.com/mtsn-multiporttsn-switch-ip-core/

[50] Lubuntu main page, [Online] Available: https://lubuntu.net/

[51] CURL main page, [Online] Available: https://curl.haxx.se/

[52] M. Bjorklund,”Yang model for Network Interfaces management”, RFC 7223, [Online] Available: https://tools.ietf.org/html/rfc7223

[53] Universidad de Granada, XXXIII Simposium Nacional de la Unión Científica Internacional de Radio, URSI 2018, [Online] Available: https://congresos.ugr.es/ursi2018/

[54] D.Rincón, A.Agustí, F.Castaño, “Controlador SDN de redes TSN en producción de TV”, poster selected in the I+D Bit program, Salon Bit Audiovisual IFEMA, Madrid, May 2018 [Online] Available: http://www.ifema.es/broadcast_01/

[55] MIB Wikipedia article, [Online] Available: https://en.wikipedia.org/wiki/Management_information_base

[56] Type-length-value Wikipedia article, [Online] Available: https://en.wikipedia.org/wiki/Type-length-value

[57] LLDP Wikipedia article, [Online] Available: https://en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol

Annex 1: MTSN Kit configuration and demos

Introduction

This guide's purpose is to explain how to perform the TSN demos included in the SoC-e's MTSN Zyqn Kit, a set of two hardware devices (bricks) that implement some TSN standards and can be used as an TSN adaptor, a TSN multiport switch and a TSN traffic generator. The guide starts with an explanation on what is needed to perform the demos, how to configure all the devices and software requirements. Then there is an explanation for every demo, the steps needed to see the results, and how these results translate to our theoretical knowledge of TSN.

The TSN protocols supported by SoC-e's MTSN Zyqn Kit hardware are:

- IEEE 1588, called IEEE 802.1AS-2011 (nanosecond range time synchronization).
- IEEE 802.1Qbv (Time-Aware Shaper).
- IEEE 802.1Qav. (Credit-Based Shaper).
- IEEE 802.1Qcc (CNC compatibility).

Tutorial and setup for the SoC-e TSN Demos

Required hardware and software

- The two MTSN Zynq Kit bricks and their power cords
- 2 or more Ethernet UTP wires
- A PC with a GNU/Linux installation and a list of software prerequisites
- USB A-to-B wire when using the 18.04 version of the firmware

Figure A1.1 MTSN Kit Zynq components

Network scenario

The specifics on the deployment will be explained on the TSN demo section. During this document the two devices will be named bricks 0 and 1.

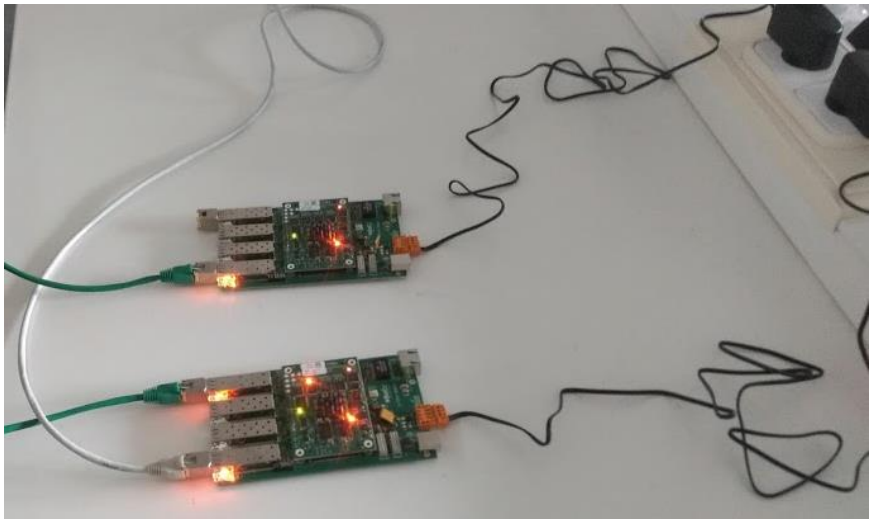


Figure A1.2 MTSN Kit Zynq deployment using firmware 16.04.3

Setup of the GNU/Linux SO

The need for a GNU/Linux OS it is related to the lack of VLAN support on Windows OS, needed by the demo.

The next section is meant to explain all the steps needed to configure and install all the prerequisites needed to perform all the tests. SoC-e provided us with an ISO with all the prerequisites installed, but preventing a future scenario where the ISO is not available, it can be mounted again using Lubuntu [50] 16.04.3 or other modern Ubuntu OS. Lubuntu is an Ubuntu distribution using the light LXDE desktop instead of the normally used GNOME, being more efficient. The PC must be connected to Internet to use the apt-get tools in order to obtain the desired files from the Ubuntu repositories.

First step - Install VideoLan (VLC) and the VLC browser plugin:

```
sudo add-apt-repository "deb http://archive.ubuntu.com/ubuntu
$(lsb release
-sc) universe"
sudo apt-get update
sudo apt-get install vlc browser-plugin-vlc
```

VideoLan will be used in the demo to receive video streams from the Zynq brick 0 and visualize them.

Second step - Create Virtual Links:

To receive video streams with different priorities, we need to create Virtual Links with different VLAN tags that differentiate the streams depending on their priority. SoC-e created a script to enable the Virtual Links. First of all we use the console command *ifconfig* to know the IP addresses of the PC we are using. Once we decide which interface is meant to be connected to the MTSN Kit network, we use it's IP address in the following steps. Then we use the script with the next command: *sudo ./createVirtualLinks.sh interfaceName*, where "interfaceName" is the interface we discovered using *ifconfig*.

The script used is copied below for the sake of changes on different implementations:

```
#!/bin/bash

ip link add link $1 name $1.11 type vlan id 1 egress 0:2 1:2 2:2
3:2 4:2 5:2 6:2 7:2
ip link add link $1 name $1.12 type vlan id 2 egress 0:5 1:5 2:5
3:5 4:5 5:5 6:5 7:5

ifconfig $1.11 192.168.5.10
ifconfig $1.12 192.168.6.10

ifconfig $1.11 up
ifconfig $1.12 up
```

The new virtual interfaces created are 192.168.5.10 and 192.168.6.10 .This script needs to be used every time the PC is restarted, as we will explain in the demo section.

Third step - Install Opera Browser:

To receive and visualize the video streams, we installed VLC and it's browser plugin. This plugin works on the Opera browser, then we need to install it via apt-get.

```
wget -O - http://deb.opera.com/archive.key | sudo apt-key add -
sh -c 'echo "deb http://deb.opera.com/opera-stable/ stable non-
free" >>
/etc/apt/sources.list.d/opera.list'
sudo apt-get update
sudo apt-get install opera
```

Fourth step - Install and configure Wireshark:

Wireshark will be the network protocol analyzer used to visualize the behavior of TSN in this demo. It can be used both to select and examine single packets and to create graphic charts representing the analysis the software is able to perform.

To install it type the nex commands:

```
sudo add-apt-repository ppa:Wireshark-dev/stable
sudo apt-get update
sudo apt-get install Wireshark
```

When using Wireshark we will need to configure the I/O graphs, this step will be explained with the rest of the demo instructions.

TSN Demo

***Important note:** Sometimes the network interface of the Ubuntu PC get deconfigured (losing the ip address) during the demos or setup, then it needs to be reconfigured using the *ifconfig* command. This is caused by the Linux network manager and it's attempts to enable DHCP and configure automatically IP addresses for our Ethernet interfaces. A solution is to stop the Linux network manager, stopping the automated services like DHCP that are not important in this demo because the network is closed and has no need of internet connection:

```
sudo systemctl stop NetworkManager.service
```

If an internet connection is needed after the demos, instead of restarting the PC the previous command can be used again changing "stop" for "start".

Once we have all the hardware and software prepared, the first step is to decide which Zynq brick will be Device 0 and which one will be Device 1 on the experiment. If we take a look at the network scenario:

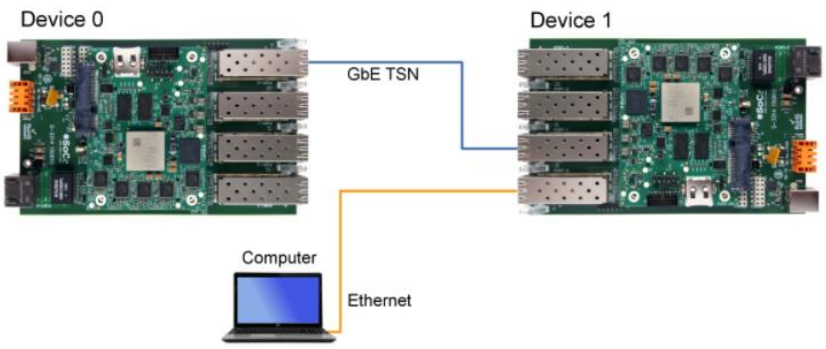


Figure A1.3 MTSN Kit deployment using firmware version 18.04

It can be seen that each brick has a different function, so we need to differentiate them before experimenting. Device0 it is the brick that sends the traffic, while Device1 it is the receiver as well as the machine that re-sends all the traffic to the Ubuntu PC, in order for the traffic to be observed using Wireshark. Connecting the Ubuntu PC to the Port0 of Device1 allows it to access the configuration tools from both devices.

The first step to is to connect via SSH to the two brick's control interface and find out what is their IP address on the interface eth1. First of all the PC with the Linux installed needs to have an interface in the same network as the two brick's Port-Z (the control interface), that is 192.168.2.X. When the interface is assigned, we connect that interface using an Ethernet UTP wire to the Port-Z of the two bricks to know their IP and decide which brick will be used as Device 0 and Device 1, as seen in Figure A1.4.

Figure A1.4 NIC usage with firmware 16.04.3

Because of some configuration changes included in the 18.04 version of the firmware, the USC connection may be needed to access the MTSN Kits internal SO, to do the same operations we will usually do using the Port-Z. To use the USB connection, plug an USB A-to-B wire between the desired brick and the Lubuntu PC, and type the next command in the console:

```
sudo minicom -s
```

This will show Minicoms menu:

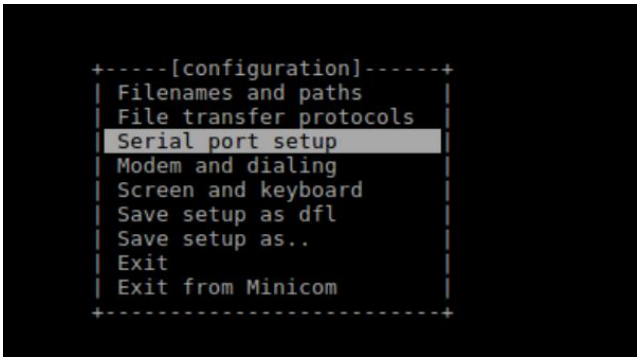


Figure A1.5 Minicom's main menu

Select Serial port setup, this will show another screen:

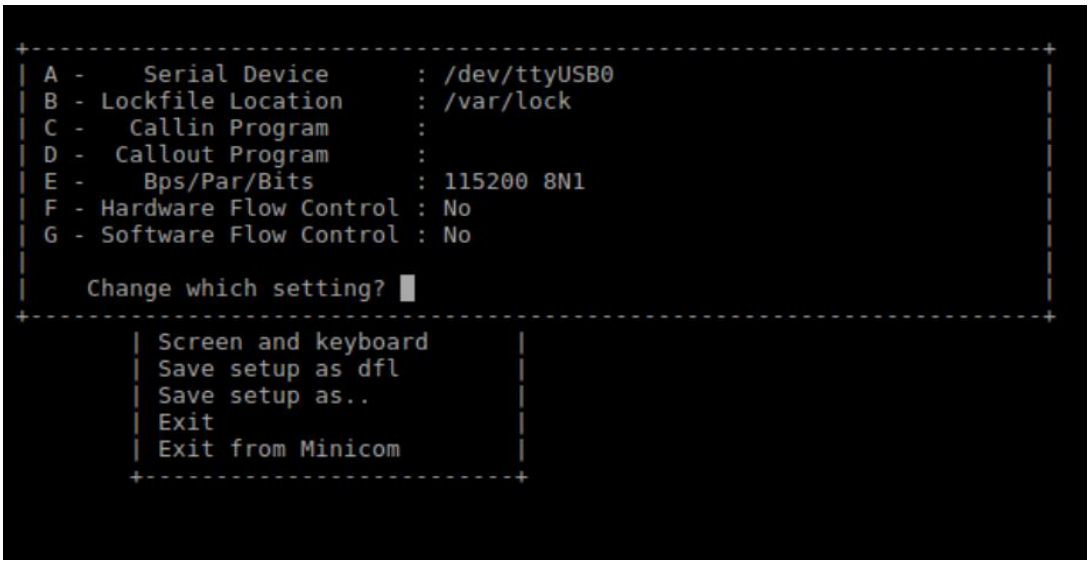


Figure A1.6 Minicom's serial port setup

To match the screen above, press *F* to set Hardware Flow Control to No, and then press *A* and type /dev/ttyUSB0. Press *Enter* to return to the first menu, then select Exit. It is important to do the command as a super user (sudo) for this connection to work. To power up the bricks, there's an AC converter that connects to the bricks using the orange plug seen in the picture. Also, before working with the bricks, there may be the need to change which interfaces will be used for the experiment. Depending on this decision, the Ethernet adapters plugged to the ports need to be reallocated.

Once the bricks are powered up and connected via Port-Z to the Linux PC,we use SSH to connect to the brick, using the console command:

```
ssh [username]@[IP]
```

By default username and password is "linaro". The default address of Port-Z is 192.168.2.64 on both bricks. Sometimes the SSH connection will be denied, showing a WARNING message with the next description: "Remote host identification has changed!" This problem is caused by previously accepting an SSH connection to a remote computer ,and that remote computer's digital fingerprint or SHA256 hash key has changed since the last connection. To get rid of the message the keys from known hosts must be deleted, using the next command:

```
sudo ssh-keygen -f "/root/.ssh/known_hosts" -R 192.168.2.64
```

Once connected, at the console, using the “sudo ifconfig” shell command, we identify the @IP of both brick’s eth1. Then we decide which brick will be working as Device 0 and Device 1, in our case we labelled the bricks to keep them identified.

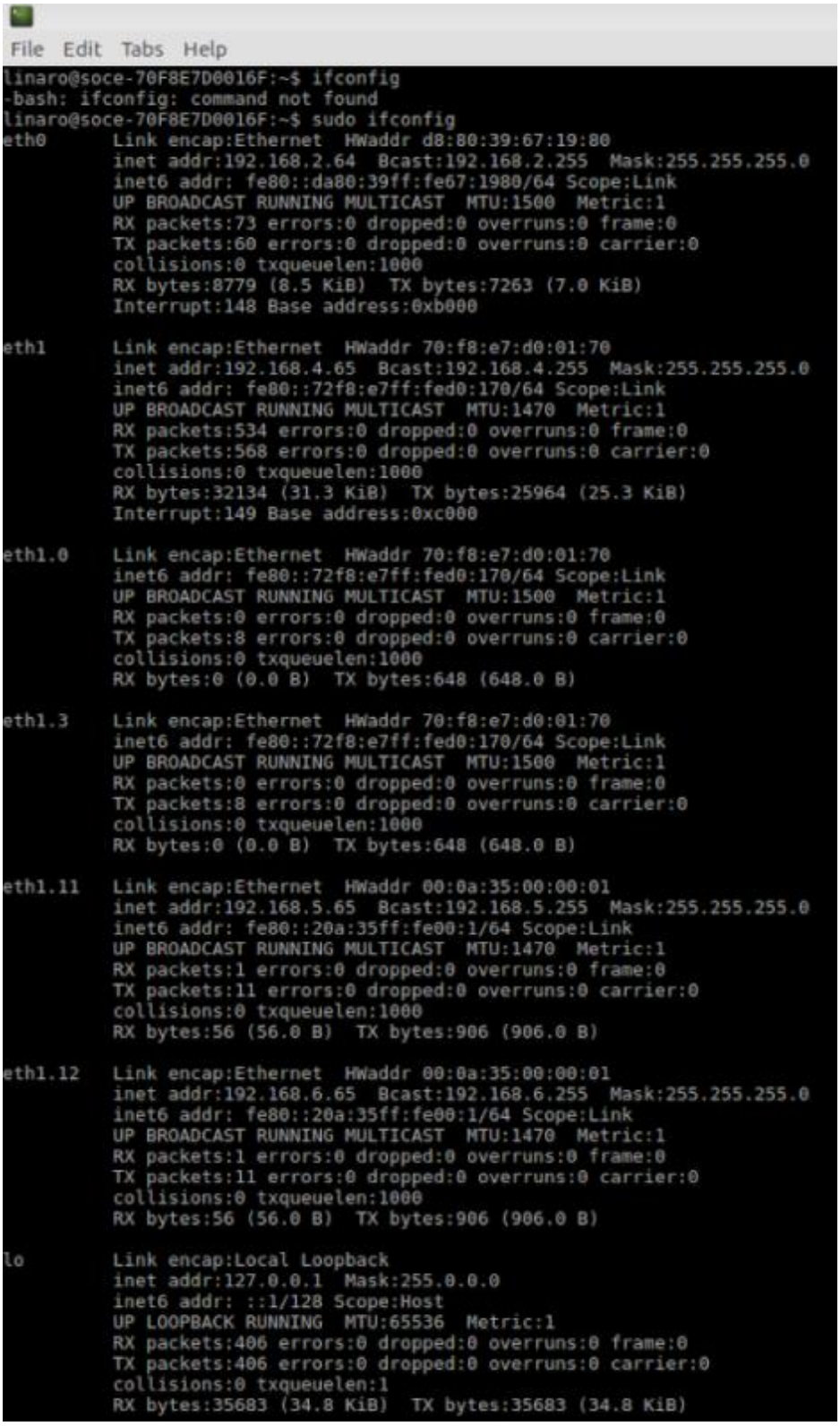


Figure A1.7 MTSN Kit default NIC configuration

We can see that when we use the ifconfig command while connected to the bricks via SSH, there is interfaces named eth1.X, these interfaces are virtual interfaces (VLAN interfaces). The addresses 192.168.5.65 and 192.168.6.65 (or 192.168.5.64 and 192.168.6.64 depending on the brick) will be used to transfer data to the virtual links created on the Ubuntu PC, while the addresses on the 192.168.4.X network will be used to access the configuration tools on the bricks.

When using the firmware version 18.04, the connection between the PC and the MTSN Kits to access the native operating system must be performed using an USB A-to-B cable. When plugged, access the terminal. With the role of each device decided, we mount the scenario shown in the picture:

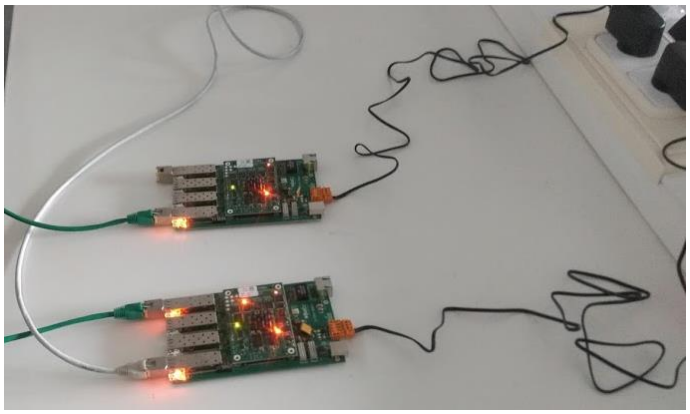


Figure A1.8 Network scenario

Due to an slightly different implementation in the hardware compared to the official documentation by the SoC-e company, we had to change the Ethernet ports connected, instead of connecting Device1 Port1 to Device0 Port3, we connected Device 1 Port3 to Device0 Port0, as can be seen on the picture the two bricks are connected by the green Ethernet wire. The white Ethernet wire connects the Linux observer PC to the Port0 of Device1.

Figure A1.9 MTSN Kit port configuration

On our Zynq bricks, only two ports are usable, being Port0 and Port3 in the current software implementation.

The next step is to connect the Ubuntu PC to the network described before, then we need to change the IP address of the connected interface to one on the 192.168.4.X group. We test the connection doing ping to both 192.168.4.64 and 192.168.4.65. Now we need access to the GUI interface to configure the device's behavior. We open the Opera browser and type "http://192.168.4.X" where X is 64 or 65 depending on which device we want to configure. An authentication pop-up will appear. The default access credentials are: username "admin" and password "SoC-e2016". The demos implement an auto-configuration tool accessed using a simple webservice, so there is no need to use the configurator tool this time.

With all the configuration and setup done, we can proceed to the demos. SoC-e provided a graphical web interface to make the access to the demo easy. To access this interface, we open the Opera browser and type http://192.168.4.X:1337, where X is the eth1 IP address from Device 0 in our scenario and 1337 is the port used by the application.

From here we have 4 demos available. This document explains all of them in the same order are presented on the website.

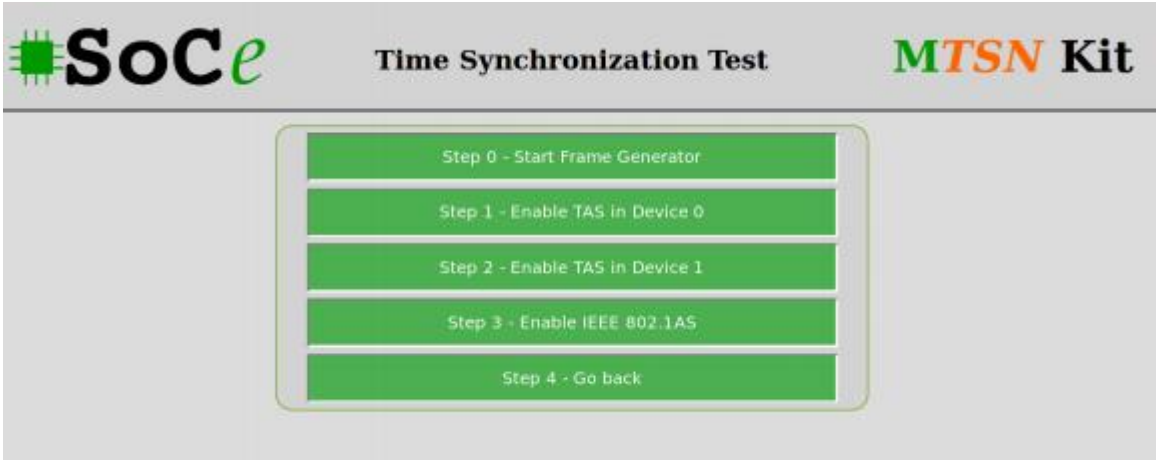


Figure A1.10 MTSN Kit demo Time Synchronization menu

Step0:Time Synchronization Test

This test demonstrates the need for a common synchronization plane in a TSN network. The division of the transmission time into cyclic windows adds the need of having a nano-second timer that allows all the devices to open these windows at the same time. The lack of this mechanism triggers a significant and random bandwidth loss.

To being able to analyze graphically the impact of using different priority traffic at the same time, we need to configure Wireshark. This step is needed for the 4

demos, but Wireshark will save the configuration so we only need to do it one time.

First we open the Wireshark software, using *sudo Wireshark* on the Linux console. We start capturing on the desired network interface. Then we select the Statistics menu, and from there the I/O Graph option. The "+" button allows us to create more input lines in the graph, we need to create 3 more and name them "vlc1", "vlc2" and "noise". The Display filter field from each one needs to be, respectively: "vlan.priority==2", "vlan.priority==5"and "vlan.priority==6". Finally we can change the color for each one, and it is advised to change the Y axis field on both three to "Bits". This way instead of seeing a packet/time graph wi will see a bits/time graph, that is more interesting to us according to the objective of the practice.

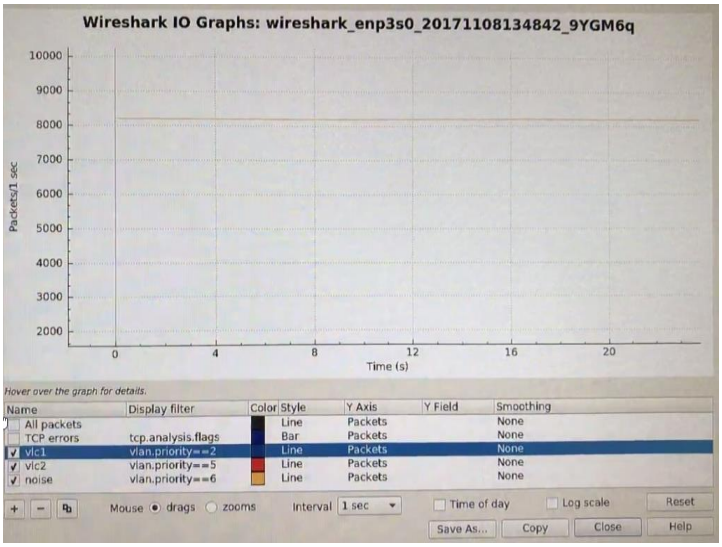


Figure A1.11 I/O graph during step 0

This will be the aspect of the I/O graph needed.

From the SoC-e demo web, click the first demo "Time Synchronization Test". Device 0 will begin transmitting data to Device 1 marked with the VLAN tag number 6. Being more concrete, the PCP(Priority code point) bits from the VLAN tag are what marks the priority of each packet.

PCP	Priority	Acronym	Traffic types
1	0 (lowest)	BK	Background
0	1(Default)	BE	Best Effort
2	2	EE	Excellent Effort
3	3	CA	Critical Applications
4	4	VI	Video, <100 ms latency and jitter
5	5	VO	Video, <10 ms latency and jitter
6	6	IC	Internetwork Control
7	7(highest)	NC	Network Control

To understand how the demo distribute traffic between priority tags, let us see the next table:

PCP	Flow
0	BE + RESTCONF + Demo
1	-
2	VLC 1
3	-
4	-
5	VLC 2
6	Noise Traffic
7	PTP Traffic(*)

(*)PTP traffic is not transmitted with the VLAN tag but MTSN IP manages this traffic as it would include a PCP value 7. Traffic sended with the 6 VLAN tag simulates noise, data that is interfering with the sending of the more important and sensitive VLC video streams.

Once we start the Step0 Demo, the next menu will popup.

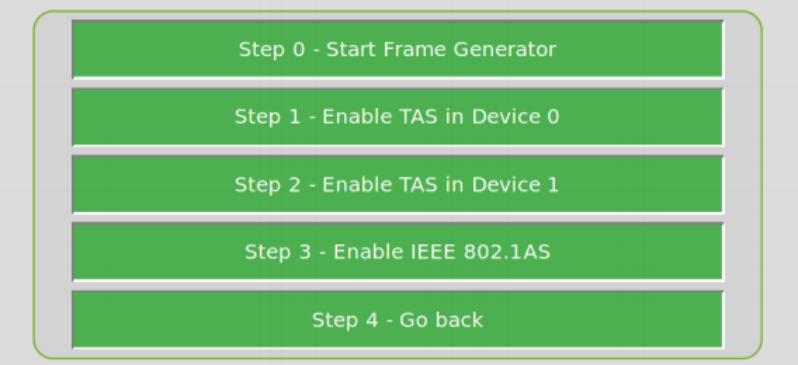


Figure A1.12 Step 0 demo menu

Click on the first step “Start Frame Generator” and start capturing with Wireshark. A transmission of around 100 Mbps is sended by the traffic generator from the Device0 to the Device1 (and re-sended to our PC), without any TSN feature enabled, and using the priority 6. The transmission uses frames of 1500 bytes and a bandwidth rate of 10%. In our first test we captured 98.448 Mbps. This “noise data” is being sended as Layer 2 packets, without more headers than the Ethernet one and the 802.1Q header to use the VLAN PCP bit priority system, the layer 2 header show that the noise data is transmitted from the MAC address associated to the frame generator and using a broadcast destination (it is sent to all the network devices in the same network). The same method to send noise traffic will be used on all the demos.

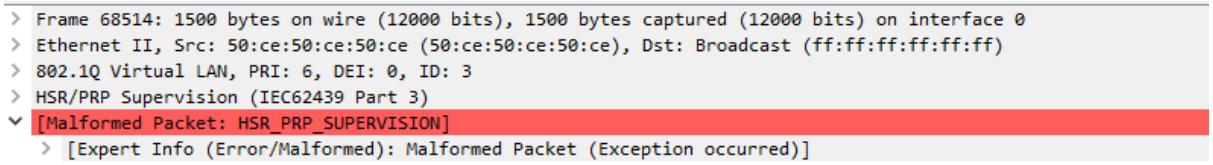


Figure A1.13 Noise data packet

Aside from the noise traffic and some HTTP messages destined to load the web resources from the server to our computer, there is some interesting messages using the Websocket protocol that send our configuration changes on the SoC-e web (in the demo associated to the steps) using the control network (192.168.4.X).It can be assumed than these messages are used to communicate with the bricks, that creates the RESTCONF messages (explained at the last step of the demo) that the SoC-e bricks will send to configure the hardware configuration. Some examples of the Websocket messages, all of them related to steps explained on the next steps of the demo:

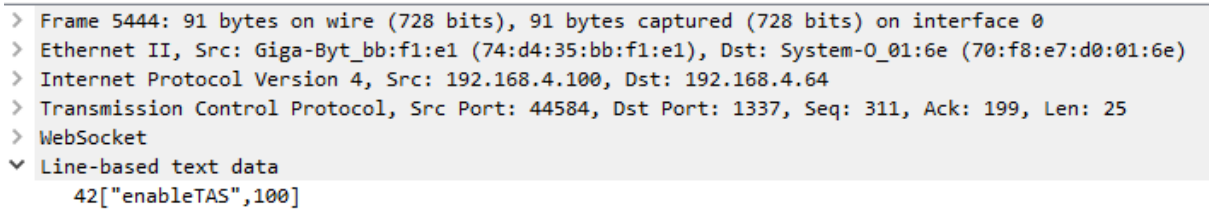


Figure A1.14 Websocket message to enable Time-Aware Shaper

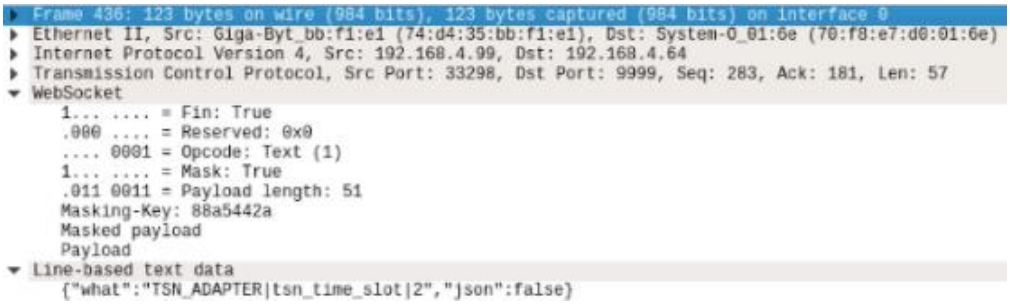


Figure A1.15 Webscoket message to designate the TAS slot number

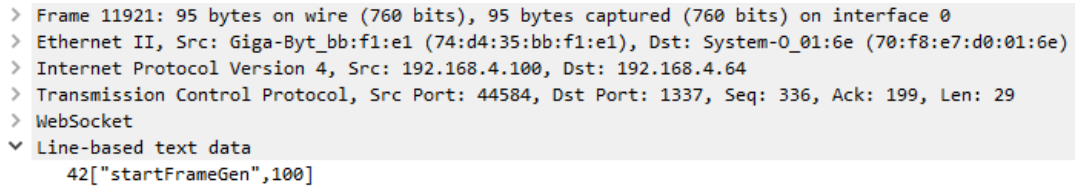


Figure A1.16 Websocket message to start the frame generator

Using the I/O graph tool can be observed the transmission of the “noise data”.

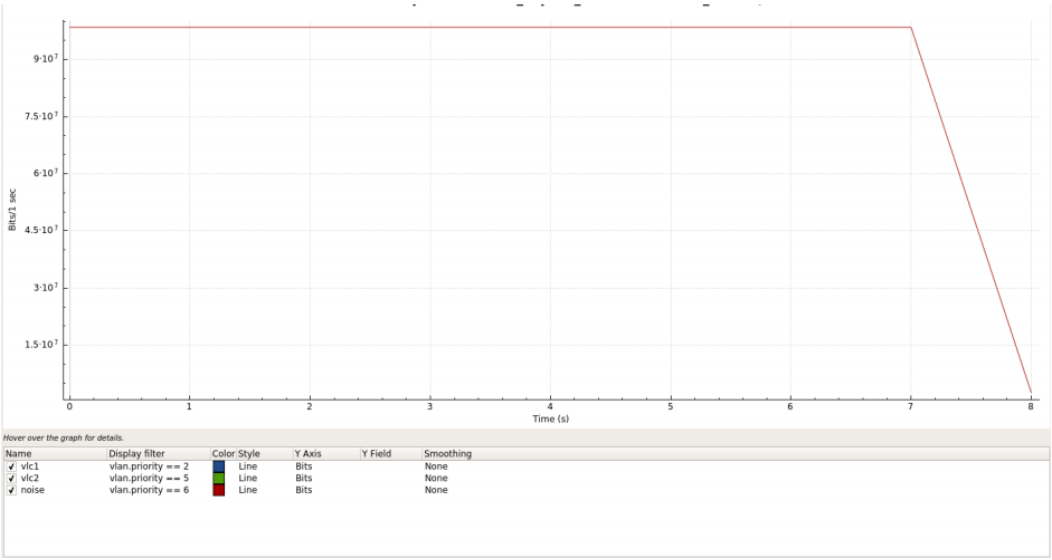


Figure A1.17 I/O graph that shows the noise data bandwidth

Click on “Enable TAS in Device 0” That enables the Time-Aware Shaper only on the Device 0.The Linux PC sends using RESTCONF the new configuration, and the brick re-configures its TSN capabilities.

The Time-Aware Shaper is configured as follows:

- Only one time slot (of the 4 available) reserved for the transmission of priority 6 traffic.
- All traffic is allowed to be transmitted in the rest of time slots.
- PTP traffic (priority 7) is allowed to be transmitted on all time slots.

The next table shows which slots can be accessed by each priority traffic:

Slot / Queue	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Slot 0	1	1	1	1	1	1	0	1
Slot 1	1	1	1	1	1	1	0	1
Slot 2	1	1	1	1	1	1	0	1
Slot 3	0	0	0	0	0	0	1	1

The expected outcome is that the transmission is being reduced to ¼ , because there is only one slot of four (and the four are equal on time) that allows the priority 6 traffic. We measured a traffic of 32 Mbps during our tests.

Figure A1.18 I/O graph after enabling the TAS on device 0

Click on “Enable TAS in Device 1”

The two bricks are non-synchronized in time, then Device 1 will not open the slots at the same time as Device 0.Depending on how big is the difference in time between when one device open the time slots compared to the other device, there will be a bigger or smaller loss of traffic.

Figure A1.19 I/O graph after enabling the TAS on both devices

We captured 7.2 Mbps, which is more than 4 times smaller than when we only had TAS active on Device0.

Figure A1.20 illustrates how a TSN network without synchronization is losing a lot of its possible potential, in this case losing bandwidth. The traffic will be only transmitted correctly during the time window when the two slots are opened at the same time.

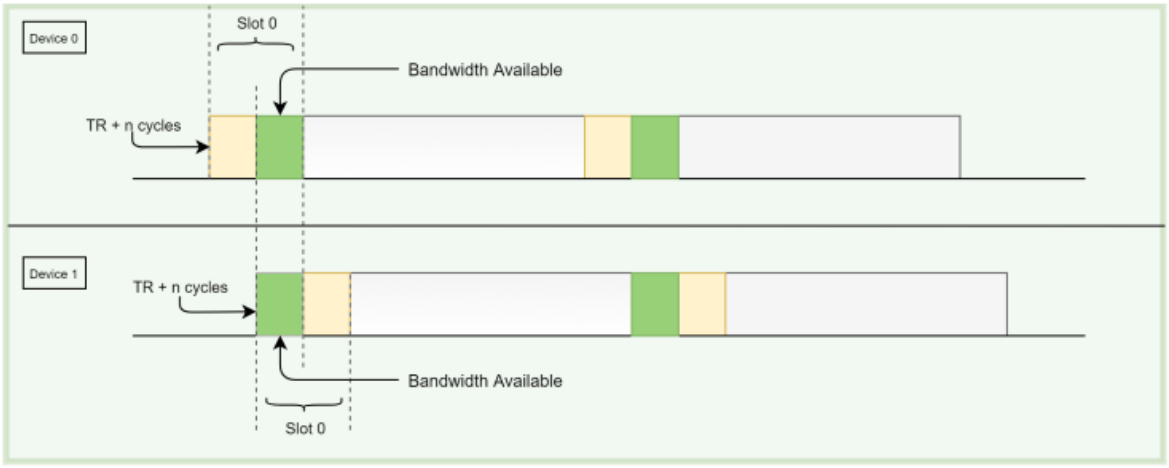


Figure A1.20 TAS enabled on both devices without PTP synchronization

Click the “Enable IEEE 802.1AS” button.

To fix the loss of bandwidth we experienced in the last step, we use IEEE 802.1AS time synchronization to make both devices open their time slots at the same time, synchronizing with precision of nanoseconds. Observing the graph it can be seen than we return to the bandwidth value we had when only Device0 used time slots, because now Device1 exactly replicates the same time slots at the same intervals.

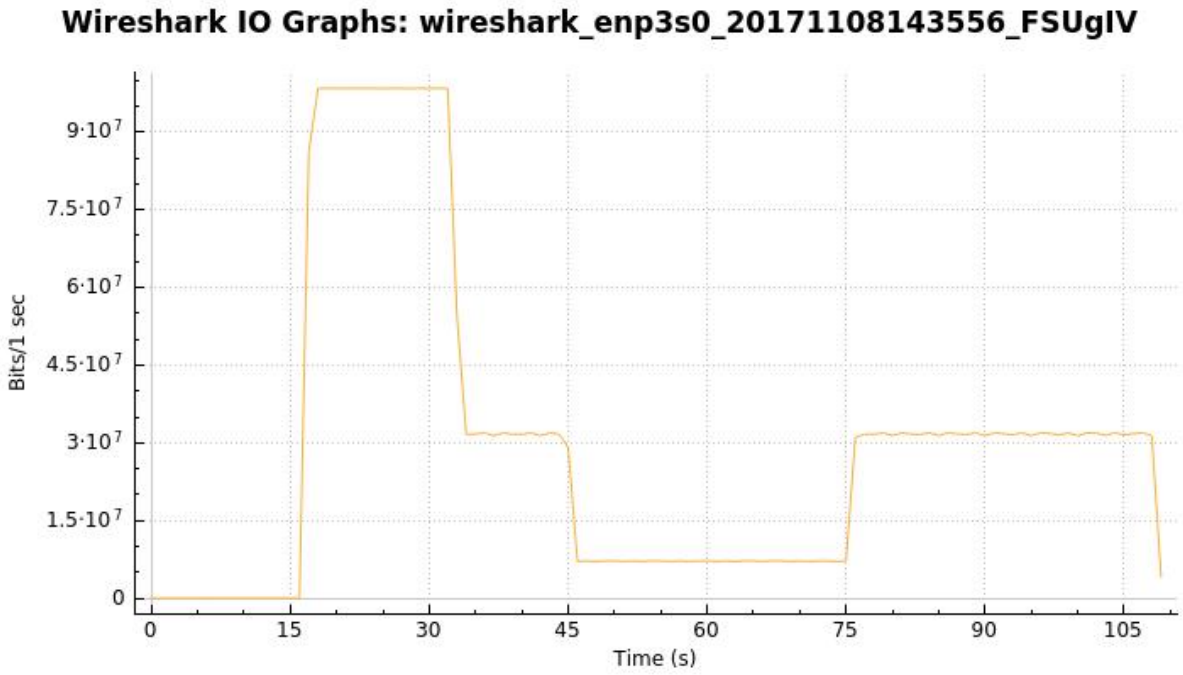


Figure A1.21 Bandwidth evolution during Step 0

Figure A1.22 shows how having time synchronization within the devices we can use all the bandwidth available in each time slot.

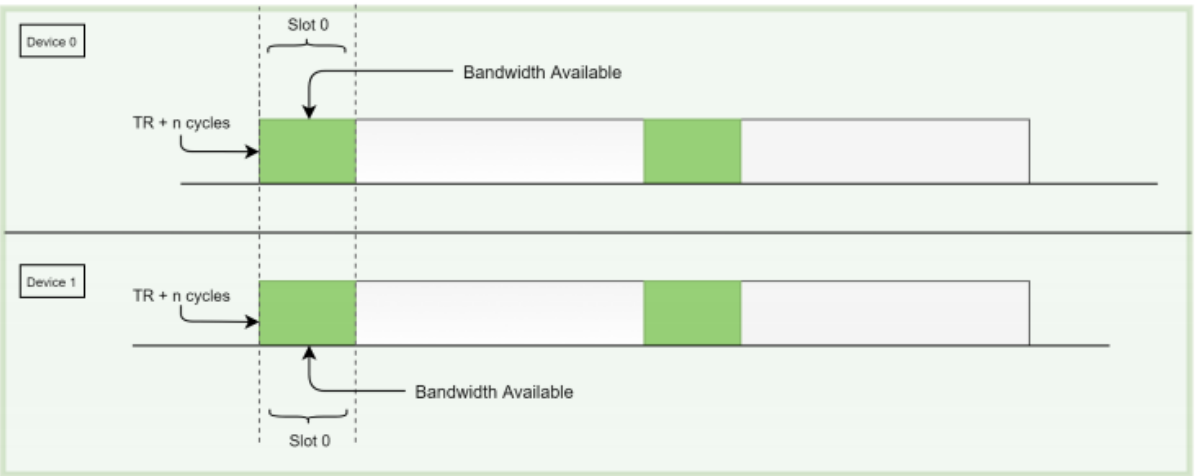


Figure A1.22 TAS enabled on both devices with PTP synchronized

Finally we click the “Go Back” button to return to the Demo selection screen, and we stop capturing in Wireshark.

Step1:Negative Test

The standard Ethernet operation is based on the Strict Priority QoS mechanism. This mechanism is based on using the PCP bits into the VLAN tag. Therefore, it is possible to assign 8 different priorities to the traffic. In a situation of congestion, the packets with less priority are being discarded. In this test, it is evidenced the impossibility of the standard Ethernet networks to perform other kind of prioritization.

To demonstrate this we will stream two VLC videos between the two bricks and we will see how interfering “noise” traffic with higher priority than the videos affect the streaming, both visually and analytically. To being able to see the videos we need to create virtual links to receive the different video streams with different priorities and use the Opera plugin to see them on the browser.

To create the VLinks we use the macro created by SoC-e:

```
sudo createVirtualLinks interfaceName
```

In our case interfaceName is enp3s0, we take the name of the interface connected to the bricks, as this doc explained on the software prerequisites section.

Note: The video streams are transmitted using the protocol HTTP which goes over TCP. This is a connection-oriented protocol which means that the initialization of the videos may be retarded the first time they are requested.

Click the Step 1 - Negative Test button to start the video transmission, and start capturing with Wireshark. If the virtual links are configured correctly, a screen like Figure A1.23 will show:

Figure A1.23 Both VLC video streams received correctly

If we did not run the script or we did it incorrectly, the two video boxes will show only a black background and the control panel.

Then the next step is to Click the "Start Frame Generator" button.

It is configured to start the traffic generator in the Device 0. The main parameters define a transmission of frames with a size of 1500 bytes with a VLAN priority of 6. The bandwidth rate is 100% in order to create a congestion situation. The two VLC instances will block, showing only the black background. This is caused by the noise traffic being higher priority, it can be seen in the Wireshark graphs that the VLC streams are not sending data because the higher priority noise traffic is using all the bandwidth. Looking at the packets transmitted, there is the noise flow described at the previous test and two TCP flows transporting the video, one from 192.168.5.64 to 192.168.5.10 and one from 192.168.6.64 to 192.168.6.10. The corresponding ACK messages follow each one of the messages from the two flows. These are the virtual interfaces created both at the Device0 and Ubuntu PC destined to transmit different priority traffic flows on one only Ethernet physical interface.

```
10788 52.233552253          192.168.6.64    192.168.6.10  TCP
10789 52.233554301          192.168.6.10    192.168.6.64  TCP
10791 52.260208938          192.168.5.64    192.168.5.10  TCP
10792 52.260221938          192.168.5.10    192.168.5.64  TCP
```

Figure A1.24 TCP video traffic

We click the “Go back” button to return to the Demo selection screen, and stop capturing with Wireshark.

Step2:Time Aware Shaper Test

As demonstrated in the previous test, with the standard operation of Ethernet networks it is impossible to prioritize the VLC instances due to the lower priorities of the flows compared to the traffic generator. In this test, it will be evidenced the TSN feature defined in the standard IEEE 802.1Qbv, which allows assigning slots of transmission for the different priorities. In particular, the parameters configured in the 802.1Qbv block will allow only the transmission of the priority 2 traffic into a single timeslot. The rest of the traffic is going to be transmitted into another timeslot. There is going to be another time slot without any traffic allowed in order to define a guard band and to improve the graphic visualization in Wireshark. The first time slot will be reserved only for the transmission of priority 0 traffic in order to keep a successful performance of the demo.

Slot / Queue	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
Slot 0	1	0	0	0	0	0	0	1
Slot 1	0	1	0	1	1	1	1	1
Slot 2	0	0	0	0	0	0	0	1
Slot 3	0	0	1	0	0	0	0	1

This test will extend what we have seen in the previous one, so we need to run the Virtual Links script if we did not before. As a consideration is important to say that the two devices are time synchronized via PTP.

Clicking the “Step 2 - Time Aware Shaper Test” will open a screen similar to the one in Step 1 (Figure A1.23).

VLC instance 1 use corresponds to the flow with VLAN priority of 2 and VLC instance 2 corresponds to the flow with VLAN priority 5. Clicking “Enable Time Aware Shaper” will configure using RESTCONF the parameters of the Time Aware Shaper on both devices. As can be observed on the previous table, VLAN priority 2 (Q2) it is the only user (with Q7 being for PTP an exception) of the Slot 3, so it has a guaranteed percentage of the bandwidth available. Then click the “Start Frame Generator” button. It is configured the same way as it was on the previous demo, to create a situation of congestion. As VLC1 flow has it’s own reserved time slot, and the bandwidth it has is enough to transmit the video, the left video will keep being available for the users to see, while the right one (VLC2) that is being transported in a flow with priority 5 that shares the same timeslot with the noise traffic will stop. For the same reason as in the previous demo (being that priority 6 it is more important than priority 5), we only lose the video flow that has no dedicated time slot and it is sharing the slot with another flow with bigger priority.

We start a capture with Wireshark for several seconds, and open the I/O graph.

Setting the X-axis scale in milliseconds we can observe how the VLC1 flow (priority 2) is being send on periodic time slots, the same as the noise priority 6 traffic.

While the VLC1 flow only sends data when needed, the slot used by the noise traffic is always using all the bandwidth it has available, as the traffic generator is configured this way.

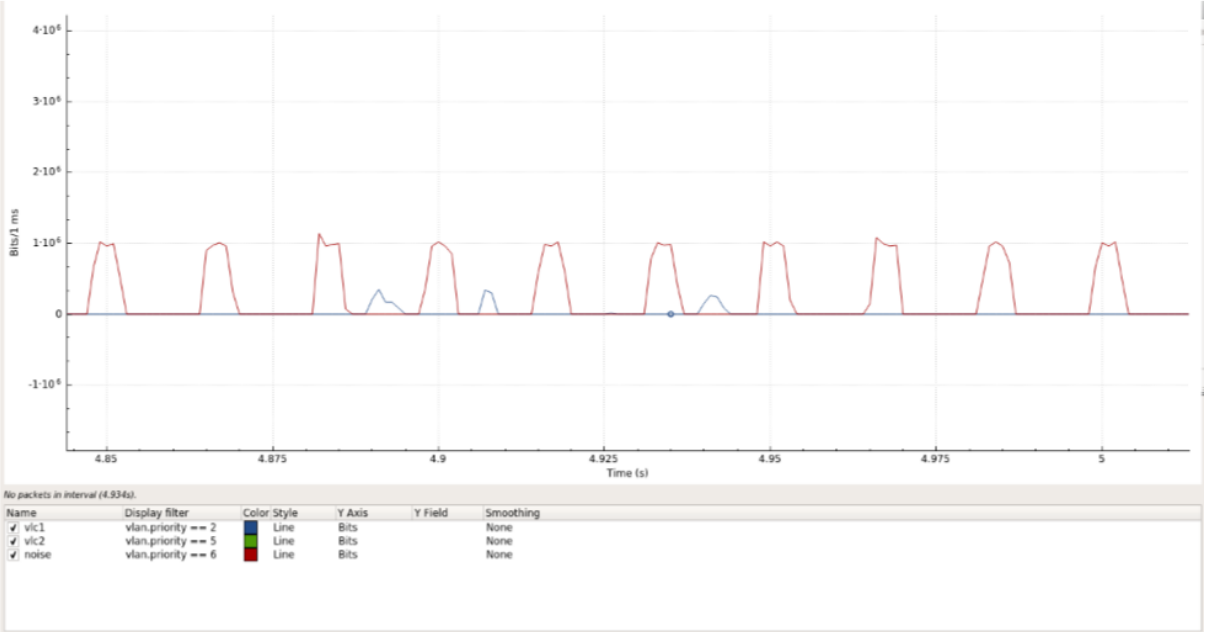


Figure A1.25 I/O graph showing both VLC flow 1 and Noise traffic

Observing the packets transmitted on Wireshark, two flows can be seen, the same ones being transmitted at the previous demo.

We click the “Go back” button to return to the Demo selection screen, and stop capturing with Wireshark.

Step3:Credit Based Shaper Test

With the Time Aware Shaper mechanism the prioritization of the priority 2 traffic has been achieved thanks to the reservation of one time slot into each transmission window. In this test the Credit-Based Shaper mechanism will be used in order to distribute the bandwidth between the traffic of several priorities. In particular, it is needed to assign enough bandwidth to the priority 5 traffic (VLC Instance 2) and to limit the bandwidth of the priority 6 traffic to a value that allows the transmission of the rest of the traffic. The values chosen are a limitation to 20% of the bandwidth to the priority 6 traffic.

This test will extend what we have seen in the previous one, so we need to run the Virtual Links script if we did not do it before. As considerations, the Time Aware Shaper is already configured exactly as in the previous test, and the

values for the Credit-Based Shaper in both devices must be the same and are already defined to reduce the complication of the test.

Clicking the “Step 2: Credit-Based Shaper Test” button will open a screen similar to the one in the two previous tests(Figure). Like in the previous tests, two videos can be seen from the browser, VLC1 with VLAN priority 2 and VLC2 with VLAN priority 5.

Clicking the “Enable Credit-Based Shaper” button will cause RESTCONF to configure on both bricks the Credit-Based Shaper parameters. In this demo, the traffic of priority 5 has a percentage of the bandwidth reserved because the limitation (20% of the total traffic) to the priority 6 traffic.

We click the “Start Frame Generator” button, that is configured to create congestion exactly like it was in the two previous tests. This time it can be seen that the two streams are received properly, thanks to limiting the traffic of priority 6.

Like in the previous tests, we will start a capture of Wireshark to observe graphically the test. This time the X-axis needs to be scaled in seconds and milliseconds to fully both the use of bandwidth and the time slots.

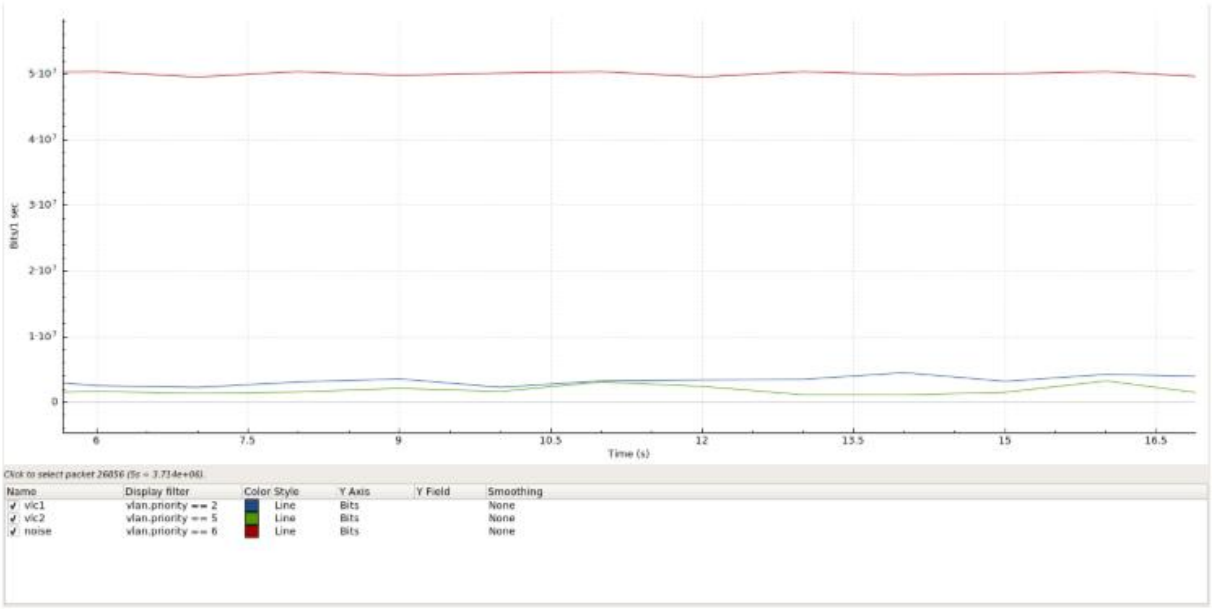


Figure A1.26 I/O Graph showing both VLC flow 1 and 2 and Noise traffic

The traffic of priority 6 now limited to 20% of the total of 250 Mbps each slot have should be around 50 Mbps, while the traffics of priorities 2 and 5 should be smaller but existent.

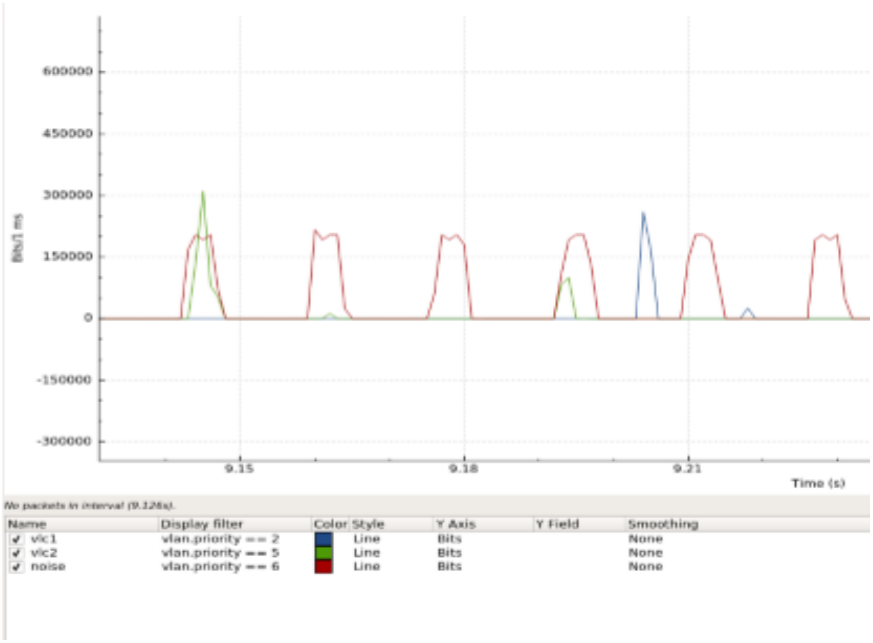


Figure A1.27 I/O graph showing both VLC flow 1 and 2 and Noise traffic

On a smaller scale the transmissions of each timeslot can be clearly be observed, like on the other experiments the priority 6 traffic is present on each Timeslot 1, but sharing this time with priority 5 traffic on the same timeslot when needed.

Step4:Centralized Configuration

To understand how the Web interface used in the demos configures the two hardware devices, take a look at the next picture:

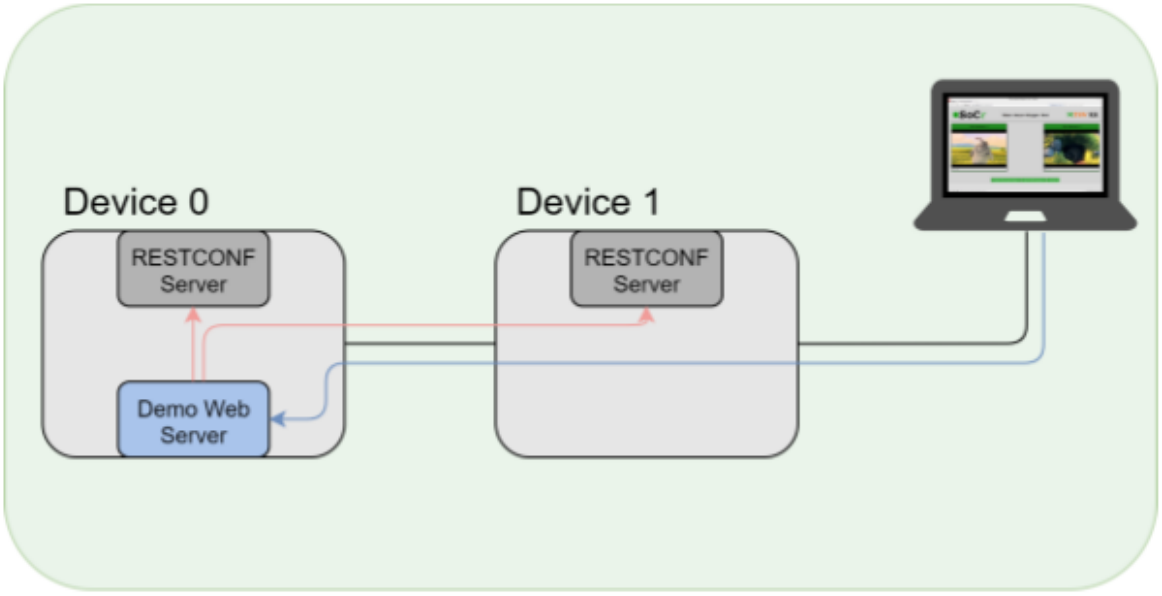


Figure A1.28 Centralized configuration topology by SoC-e

The Linux PC is connected to the Device0, that hosts the web server.

The web server sends HTTP requests to both device's RESTCONF servers in order to configure them depending on the user decisions using the web interface or the CLI. RESTCONF is a protocol created to obtain the capabilities from any network hardware and be able to configure its configuration, using HTTP requests (GET,POST,DELETE…) and responses to obtain and deliver the data.

Both requests and responses can be in format JSON or XML, but they need to follow a YANG data model. This model is updated to include schedules for every IEEE 802.1 standard, thanks to that the hardware developers can made their devices configurable using any RESTCONF client, like the web from the demo.

The YANG model for 802.1Qbv is fully described in [21], being the only TSN protocol configurable via YANG models in this version (17.06) of the MTSN Kit software. In the current version of the software running on the bricks is impossible to observe the content of the HTTP RESTCONF messages (HTTP2 on this implementation), because they are transmitted using over a SSL secure connection, and to decrypt it with Wireshark the encryption key is needed, and is not available in the current release. The same key is needed to send HTTP configuration request/response messages using Postman. The SoC-e team is planning the including of a tool for making this centralized configuration as a RESTCONF client with a user friendly graphical interface. It would be present on future releases.

Commentary 1: PTP

During the demos we can see the Path Delay Request Messages, shown in the next picture:

```
▾ Precision Time Protocol (IEEE1588)
  › 0001 .... = transportSpecific: 0x1
    .... 0010 = messageId: Path_Delay_Req Message (0x2)
    .... 0010 = versionPTP: 2
    messageLength: 54
    subdomainNumber: 0
  ▾ flags: 0x0000
      0... .... .... .... = PTP_SECURITY: False
      .0.. .... .... .... = PTP profile Specific 2: False
      ..0. .... .... .... = PTP profile Specific 1: False
      .... .0.. .... .... = PTP_UNICAST: False
      .... ..0. .... .... = PTP_TWO_STEP: False
      .... ...0 .... .... = PTP_ALTERNATE_MASTER: False
      .... .... ..0. .... = FREQUENCY_TRACEABLE: False
      .... .... ...0 .... = TIME_TRACEABLE: False
      .... .... .... 0... = PTP_TIMESCALE: False
      .... .... .... .0.. = PTP_UTC_REASONABLE: False
      .... .... .... ..0. = PTP_LI_59: False
      .... .... .... ...0 = PTP_LI_61: False
  › correction: 0.000000 nanoseconds
    ClockIdentity: 0x70f8e7fffed00170
    SourcePortID: 1
    sequenceId: 78
    control: Other Message (5)
    logMessagePeriod: 0
```

Figure A1.29 PTP Path Delay Request Message

The function of these messages is to know the time delay between an slave clock and the master clock to correct it. There is no Path Delay Responses observable though. The PTP clock negotiation is already preconfigured in the devices, so we cannot observe how they negotiate and assign the clocks.

Commentary 2: VLAN Priorities

Observing the frames with the Wireshark tool, the three traffic flow frames have associated a 802.1Q VLAN header, as explained during the demo. An example of each one is provided here to illustrate how these headers are showed, what fields they include, and to which VLAN ID each priority is associated. Note that every priority is accompanied by a textual description of its use on a normal VLAN network (Voice, Control, Excellent Effort).

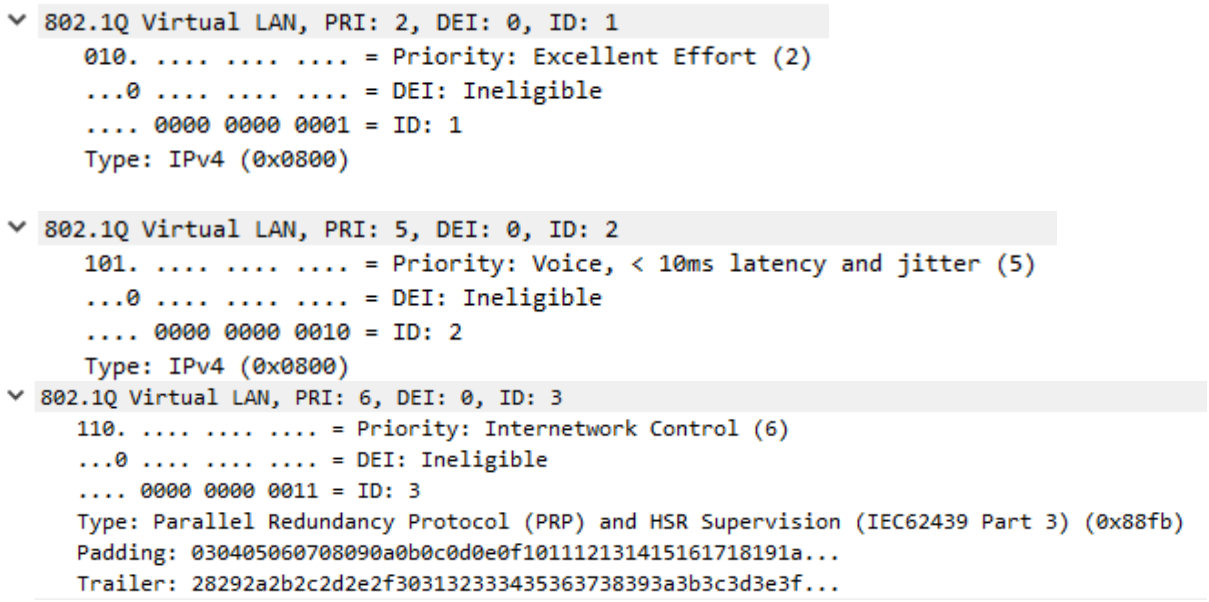


Figure A1.30 VLAN PCP Tags

Annex 2: Usage of CURL

The CURL [51] tool is used to create the HTTP messages used to interact with the Jetconf server hosted by the MTSN Kits. When using that tool in the lab, some necessary preliminary steps must be performed, most of them related to how Jetconf processes petitions. The next textbox shows a list of commands to make easier the usage in any Ubuntu operating system, included the Lubuntu ISO provided by SoC-e to use with their MTSN Kits.

```
sudo apt-get install build-essential nghttp2 libnghttp2-dev wget
https://curl.haxx.se/download/curl-7.54.0.tar.bz2

tar -xvjf curl-7.54.0.tar.bz2

cd curl-7.54.0

./configure --with-nghttp2 --prefix=/usr/local

make

sudo make install

sudo ldconfig

sudo apt-get install libssl-dev

./configure --with-nghttp2 --prefix=/usr/local --with-
ssl=/usr/local/ssl

sudo ./configure --with-darwinssl

make

sudo make install
```

Annex 3: YANG Module for the TAS

This annex shows how the “gate-parameters” block from the YANG module developed to describe the time-aware is organized.

```
container gate-parameters {
      description
"A list that contains the per-port managable parameters for traffic
scheduling. For a given Port, an entry in the table exists. All
writable objects in this table must be persistent over power up
restart/reboot.";
      reference
        "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9, 12.29.1";

      leaf gate-enabled {
        type boolean;
        description
"The GateEnabled parameter determines whether traffic scheduling is
active (true) or inactive (false).The value must be retained across
reinitializations of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.2, 8.6.9.4.14, 12.29.1";
      }
      leaf admin-gate-states {
        type uint8;
        description
"AdminGateStates is the administrative value of the initial gate
states for the Port. The bits of the octet represent the gate states
for the corresponding traffic classes; the most-significant bit
corresponds to traffic class 7, the least-significant bit
to traffic class 0. A bit value of 0 indicates closed; a bit value of
1 indicates open. The value must be retained across reinitializations
of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.5, 12.29.1";
      }
      leaf oper-gate-states {
        type uint8;
        description
      }
      leaf admin-control-list-length {
        type uint32;
        description
"AdminControlListLength is the number of entries in the
AdminControlList. The value must be retained across reinitializations
of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.6, 12.29.1.2";
      }
      leaf oper-control-list-length {
        type uint32;
        description
   reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.23, 12.29.1.2";
      }

 list admin-control-list {
```

```
        key "index";
        description
"AdminControlList is the administrative value of the gate control list
for the Port. The value must be retained across reinitializations of
the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9.4.2, 12.29.1.2";

        leaf index {
          type uint32;
          description
"This index is provided in order to provide a unique key per list
entry. The value of index for each entry shall be unique (but not
necessarily contiguous).";
        }

        uses gate-control-entry;
      }
      list oper-control-list {
        key "index";
        description
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9.4.19, 12.29.1.2";

        leaf index {
          type uint32;
          description
"This index is provided in order to provide a unique key per list
entry. The value of index for each entry shall be unique(but not
necessarily contiguous).";
        }

        uses gate-control-entry;
      }
      container admin-cycle-time {
        description
          "AdminCycleTime specifies the administrative value
          of the gating cycle time for the Port.
           AdminCycleTime is a rational number of seconds,
          defined by an integer numerator and an integer
          denominator.
          The value must be retained across
          reinitializations of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9.4.3, 12.29.1";
        leaf numerator {
          type uint32;
          description "AdminCycleTime's numerator.";
        }
        leaf denominator {
          type uint32;
          description "AdminCycleTime's denominator.";
        }
      }
      container oper-cycle-time {
        description
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.8.4, 8.6.9.4.20, 12.29.1";
        leaf numerator {
          type uint32;
          description "OperCycleTime's numerator.";
```

```
        }
        leaf denominator {
          type uint32;
          description "OperCycleTime's denominator.";
        }
      }
      leaf admin-cycle-time-extension {
        type uint32;
        description
          "An unsigned integer number of nanoseconds,
          defining the maximum amount of time by which the gating
          cycle for the Port is permitted to be extended when a
          new cycle configuration is being installed. This is the
          administrative value.
          The value must be retained across
          reinitializations of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.4, 12.29.1";
      }
      leaf oper-cycle-time-extension {
        type uint32;
        description
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.21, 12.29.1";
      }
      container admin-base-time {
        description
"The administrative value of the base time at which gating cycles
begin, expressed as an IEEE 1588 precision time protocol (PTP)
timescale. The value must be retained across re- initializations of
the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.1, 12.29.1";
        uses ptp-timestamp;
      }
      container oper-base-time {
        description
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.18, 12.29.1";
        uses ptp-timestamp;
      }
      leaf config-change {
        type boolean;
        description
"The ConfigChange parameter signals the start of a configuration
change when it is set to TRUE, indicating that the administrative
parameters for the Port are ready to be copied into their
corresponding operational parameters. This should only be done when
the various administrative parameters are all set to appropriate
values.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.7, 12.29.1";
      }
      container config-change-time {
        description
"The time at which the next config change is scheduled to occur.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.9, 12.29.1";
        uses ptp-timestamp;
      }
      leaf tick-granularity {
```

```
        type uint32;
        description
"The granularity of the cycle time clock, represented as an unsigned
number of tenths of nanoseconds. The value must be retained across
reinitializations of the management system.";
        reference
          "IEEE Std 802.1Qbv-2015: 12.29.1";
      }
      container current-time {
        description
          "The current time as maintained by the local system.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.10, 12.29.1";
        uses ptp-timestamp;
      }
      leaf config-pending {
        type boolean;
        description
"The value of the ConfigPending state machine variable.
The value is TRUE if a configuration change is in progress but has not
yet completed.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.8, 12.29.1";
      }
      leaf config-change-error {
        type yang:counter64;
        description
"A counter of the number of times that a re-configuration of the
traffic schedule has been requested with the old schedule still
running and the requested base time was in the past.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.3.1, 12.29.1";
      }
      leaf supported-list-max {
        type uint32;
        description
"The maximum value supported by this Port for the
AdminControlListLength and OperControlListLength parameters. It is
available for use by schedule computation software to determine the
port’s control list capacity prior to computation.";
        reference
          "IEEE Std 802.1Qbv-2015: 8.6.9.4.21, 12.29.1.5";
      }
```

Annex 4: RESTCONF HTTP messages

This annex shows the next RESTCONF operations, providing examples for a better understanding of the protocol:

- Request to start the configuration process, followed by a confirmation.

```
---------------------------------REQUEST-------------------------

POST https://192.168.4.65:8443/restconf/operations/jetconf:conf-start


-----------------Data-----------------
{"jetconf:input":{"name":"802.1Qbv edit"}}
-------------------------------------

---------------------------------RESPONSE-----------------------
--------------Headers----------------
:status: 200
content-type: application/yang.api+json
content-length: 22
server: jetconf-h2
cache-control: No-Cache
access-control-allow-origin: all
access-control-allow-headers: Content-Type
-------------------------------------

-----------------Data-----------------
{
    "status": "OK"
}
-------------------------------------
```

- Request to change the configuration of an interface, setting 3 time slots in the Time-Aware Shaper as well as the allowed VLAN queues in each slot.

```
---REQUEST-------------------------
PUT                      https://192.168.4.65:8443/restconf/data/ietf-
interfaces:interfaces/interface=0/ieee802-dot1q-bridge:bridge-port


-----------------Data-----------------
{"ieee802-dot1q-bridge:bridge-port":{"ieee802-dot1q-sched:gate-
parameters":{"admin-base-time":{"fractional-
seconds":0,"seconds":0},"admin-cycle-
time":{"denominator":41666.666666666664,"numerator":500},"config-
change":true,"admin-control-list-length":3,"admin-control-
list":[{"index":0,"sgs-params":{"gate-states-value":7,"time-interval-
value":4000000}},{"index":1,"sgs-params":{"gate-states-
value":56,"time-interval-value":4000000}},{"index":2,"sgs-
params":{"gate-states-value":224,"time-interval-value":4000000}}]}}}


---RESPONSE-----------------------
```

```
---------------Headers-----------------
:status: 204
content-type: text/plain
server: jetconf-h2
cache-control: No-Cache
access-control-allow-origin: all
access-control-allow-headers: Content-Type
---------------------------------------

-----------------Data------------------
```

- The next operation is the same as the previous one but on the other Ethernet port (bridge-port:1).

```
REQUEST------------------------
PUT                      https://192.168.4.65:8443/restconf/data/ietf-
interfaces:interfaces/interface=1/ieee802-dot1q-bridge:bridge-port


-----------------Data------------------
{"ieee802-dot1q-bridge:bridge-port":{"ieee802-dot1q-sched:gate-
parameters":{"admin-base-time":{"fractional-
seconds":0,"seconds":0},"admin-cycle-
time":{"denominator":41666.666666666664,"numerator":500},"config-
change":true,"admin-control-list-length":3,"admin-control-
list":[{"index":0,"sgs-params":{"gate-states-value":7,"time-interval-
value":4000000}},{"index":1,"sgs-params":{"gate-states-
value":56,"time-interval-value":4000000}},{"index":2,"sgs-
params":{"gate-states-value":224,"time-interval-value":4000000}}]}}}
---------------------------------------

---------------------------------RESPONSE-----------------------
---------------Headers-----------------
:status: 204
content-type: text/plain
server: jetconf-h2
cache-control: No-Cache
access-control-allow-origin: all
access-control-allow-headers: Content-Type
---------------------------------------

-----------------Data------------------
```

- Operation to commit the changes, followed by a confirmation response.

```
---REQUEST-------------------------
POST https://192.168.4.65:8443/restconf/operations/jetconf:conf-commit

-----------------Data------------------


---RESPONSE------------------------
---------------Headers-----------------
:status: 200
content-type: application/yang.api+json
content-length: 48
server: jetconf-h2
cache-control: No-Cache
```

```
access-control-allow-origin: all
access-control-allow-headers: Content-Type
--------------------------------------

-----------------Data-----------------
{
    "conf-changed": true,
    "status": "OK"
}
--------------------------------------
```

- The last operation is a request to retrieve the Time-Aware Shaper information for all the interfaces in the MTSN Kit. The response message shows the configuration in the same format already explained in this document.

```
---REQUEST-------------------------
GET https://192.168.4.65:8443/restconf/data/ietf-interfaces:interfaces


-----------------Data-----------------

---RESPONSE------------------------
---------------Headers----------------
:status: 200
content-type: application/yang.api+json
content-length: 7703
server: jetconf-h2
cache-control: No-Cache
access-control-allow-origin: all
access-control-allow-headers: Content-Type
etag: -1693021603
last-modified: Sat, 21 May 2016 22:34:33 GMT
--------------------------------------

-----------------Data-----------------
{
    "ietf-interfaces:interfaces": {
        "interface": [
            {
                "enabled": true,
                "type": "EthernetCsmacd",
                "ieee802-dot1q-bridge:bridge-port": {
                    "ieee802-dot1q-sched:gate-parameters": {
                        "admin-cycle-time": {
                            "denominator": 41666,
                            "numerator": 500
                        },
                        "admin-control-list-length": 3,
                        "admin-control-list": [
                            {
                                "index": 0,
                                "sgs-params": {
                                    "gate-states-value": 7,
                                    "time-interval-value": 4000000
                                }
                            },
                            {
```

```
                                "index": 1,
                                "sgs-params": {
                                    "gate-states-value": 56,
                                    "time-interval-value": 4000000
                                }
                            },
                            {
                                "index": 2,
                                "sgs-params": {
                                    "gate-states-value": 224,
                                    "time-interval-value": 4000000
                                }
                            }
                        ],
                        "admin-base-time": {
                            "fractional-seconds": "0",
                            "seconds": "0"
                        },
                        "config-change": true
                    }
                },
                "name": "0"
            },
            {
                "enabled": true,
                "type": "EthernetCsmacd",
                "ieee802-dot1q-bridge:bridge-port": {
                    "ieee802-dot1q-sched:gate-parameters": {
                        "admin-cycle-time": {
                            "denominator": 41666,
                            "numerator": 500
                        },
                        "admin-control-list-length": 3,
                        "admin-control-list": [
                            {
                                "index": 0,
                                "sgs-params": {
                                    "gate-states-value": 7,
                                    "time-interval-value": 4000000
                                }
                            },
                            {
                                "index": 1,
                                "sgs-params": {
                                    "gate-states-value": 56,
                                    "time-interval-value": 4000000
                                }
                            },
                            {
                                "index": 2,
                                "sgs-params": {
                                    "gate-states-value": 224,
                                    "time-interval-value": 4000000
                                }
                            }
                        ],
                        "admin-base-time": {
                            "fractional-seconds": "0",
                            "seconds": "0"
                        },
                        "config-change": true
```

```
                    }
                },
                "name": "1"
            },
            {
                "enabled": true,
                "type": "EthernetCsmacd",
                "ieee802-dot1q-bridge:bridge-port": {
                    "ieee802-dot1q-sched:gate-parameters": {
                        "admin-cycle-time": {
                            "denominator": 50000,
                            "numerator": 500
                        },
                        "admin-control-list-length": 4,
                        "admin-control-list": [
                            {
                                "index": 0,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 1,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 2,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 3,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            }
                        ],
                        "admin-base-time": {
                            "fractional-seconds": "0",
                            "seconds": "0"
                        },
                        "config-change": true
                    }
                },
                "name": "2"
            },
            {
                "enabled": true,
                "type": "EthernetCsmacd",
                "ieee802-dot1q-bridge:bridge-port": {
                    "ieee802-dot1q-sched:gate-parameters": {
                        "admin-cycle-time": {
                            "denominator": 50000,
                            "numerator": 500
```

```
                        },
                        "admin-control-list-length": 4,
                        "admin-control-list": [
                            {
                                "index": 0,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 1,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 2,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            },
                            {
                                "index": 3,
                                "sgs-params": {
                                    "gate-states-value": 255,
                                    "time-interval-value": 2500000
                                }
                            }
                        ],
                        "admin-base-time": {
                            "fractional-seconds": "0",
                            "seconds": "0"
                        },
                        "config-change": true
                    }
                },
                "name": "3"
            }
        ]
    }}
```

Annex 5: MTSN Kit overview

Multiport TSN (MTSN) [19] is a solution for any customer that requires an all-in-one solution to introduce Time Sensitive Networking in their equipment. It is based on a FPGA and works both as a TSN Switch, configured graphically using the Xilling Vivado Tool. There are two versions available, depending on theSoM (system on module) used, MPSoC and Zynq.

MTSN Kit MPSoC, featuring 6 Ethernet ports.

Figure A5.1 MTSN Kit MPSoC

MTSN Kit Zynq, featuring 2 Ethernet ports.

Figure A5.2 MTSN Kit Zynq

The block diagram of the design is as follows, only the MPSoC version is shown to simplify this section.

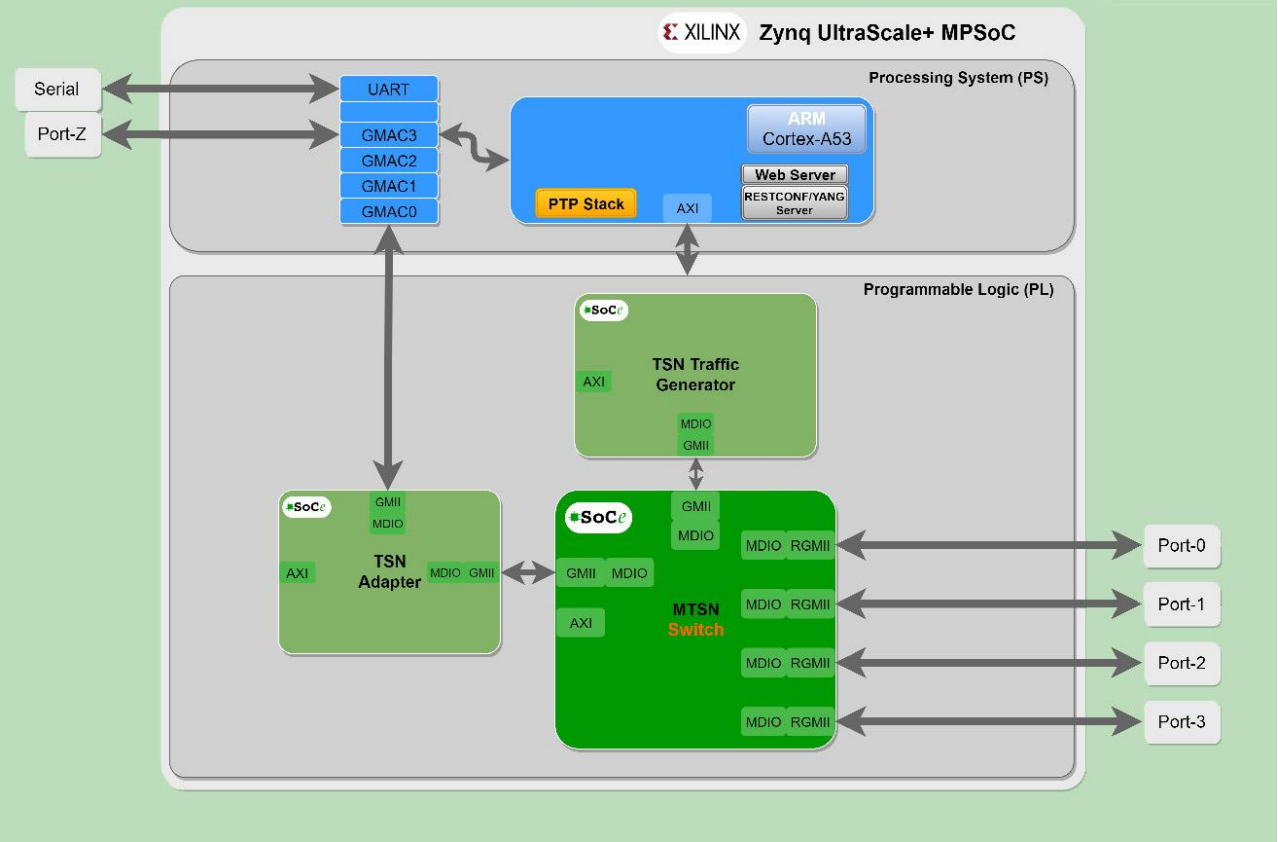


Figure A5.3 MTSN Kit design block diagram from [19]

The main elements included in this design are:

- PS Section: Processing System section of the SoC.
- PL Section: Programmable Logic section of the SoC.
- TSN Traffic Generator: HDL (hardware description language) Block able to generate different class Ethernet Traffic with controlled throughput. Additionally, it integrates a Time-Aware Shaper for scheduling the conventional Ethernet traffic generated by the traffic generator block.
- TSN Adapter: HDL Block in charge of implementing the Time-Aware Shaper for scheduling the conventional Ethernet traffic managed by the standard regular Ethernet MAC of the Processor Section of the SoC.
- MTSN Switch IP: Six or two ports TSN switch. These ports are distributed as two internal and four external ports.

Implementation of TSN standards in the MTSN Kits

The MTSN Kit is designed to be compatible with the main TSN standards. Even so, the product is evolving at this moment with the goal of improve and expand its capabilities.

The current TSN standards this implementation supports are:

- Time Synchronization, based on the IEEE 802.AS-2011 protocol.
- Traffic Shaping, based on the IEEE 802.1Qbv Time-Aware Shaper and the IEEE 802.1Qav Credit-Based Shaper.
- Network Configuration, based on the IEEE 802.1Qcc centralized network configuration.

MTSN Kit TSN usage

This section details how the MTSN Kit implements the TSN functions, based on experiments performed at the laboratory. For more context on the obtained values and the reasoning behind some decisions during the experiments, Annex 1 focuses on explaining the setup and configuration for the MTSN Kits, as well as a tutorial to follow the experiments the hardware features for demonstration purposes.

PTP

PTP (Precise Time Protocol) is the protocol used to synchronize the clocks in a TSN network. The MTSN Kit implementation uses the Linux PTP protocol library ptp4l, implements master selection to negotiate which of the clocks will serve as a master to correct the other clocks on the network, as well as BC (boundary clock).

Traffic Shaping

The MTSN Kit implements both the time-aware and Credit-Based Shaper in order to assure that the priority traffic is delivered in time regardless of the transmission of other traffic streams. The Time-Aware Shaper uses the PCP (Priority Code Point) VLAN tags to separate different traffic priority and designate which traffic queue can transmit during each of the time slots the shaper separates from a transmission period.In the MTSN Kits, the shaper can separate each transmission period in 4 slots, while the total amount of PCP values is 8, from 0 to 7, having the higher values a bigger priority.

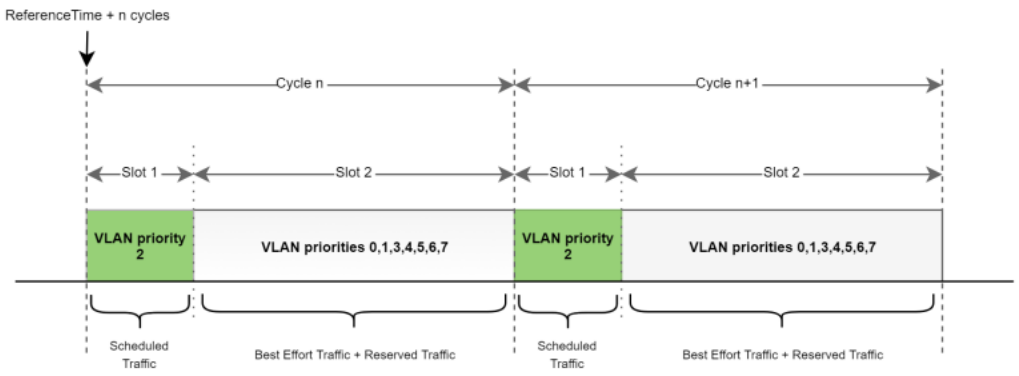


Figure A5.4 Time aware schedule

The Credit-Based Shaper assign credits to the queues with reserved bandwith, that credit is consumed with every transmission from the queue. While the sending credit is positive, data frames with reserved bandwith can be transmitted, but when the credit decreases beneath 0, it needs to wait until it is positive again. The sending credit increases on a queue if its transmission is delayed because of other transmission.

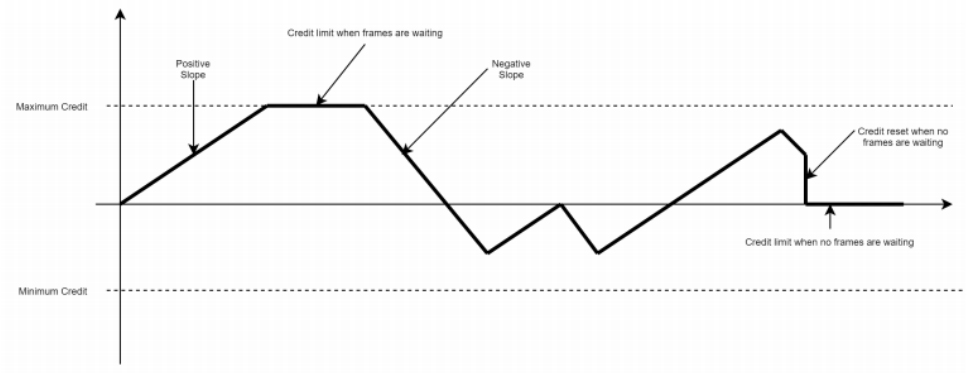


Figure A5.5 Credit based scheduler credit usage

To set the values for both schedulers, the MTSN Kit has a configuration screen that can be accessed using the webservice available to change and see its configuration parameters, as illustrated in A5.6

Time Aware Shaper	
Window Duration	1250000
Number Of Time Slots	4
TSN Time Slot	3
Time Slot Duration	312500
Allowed Queues in Time Slot	0x000000FF

Figure A5.6 Time aware shaper configuration

One of the fields that can be configured is the “Allowed Queues in Time Slot”. This field associates each time slot of the 4 available to one or more priority queues, depending on the PCP VLAN tag they have associated.

Selecting a Slot in the configuration menu (typing a value from 0 to 3 in the box) and refreshing the website show how the queues are allowed or not in the slot. The format is showed and configured using an Hexadecimal chain with 8 digits, but only the last two digits are important due to only having 8 priority queues.

As an example:
Hex – 0x00000001 –> Bin – 0b00000001 –> Only queue 0 allowed
Hex – 0x0000000F –> Bin – 0b00001111 –> Allowed queues 0,1,2,3
Hex – 0x000000FF –> Bin – 0b11111111 –> All queues allowed

To study how SoC-e’s bricks change the distribution on different scenarios, a transcription of the hexadecimal chains to an association table is an interesting exercise.

MTSN Zyqn bricks default state

All queues are allowed in all time slots.
Time slot 0,1,2 and 3
0x000000FF-->0b11111111

Slot/queue	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1
3	1	1	1	1	1	1	1	1

Time Synchronization

One of the featured demos on the MTSN Bricks configures the next Time-Aware Shaper setup, first on only one brick and finally on both, to show that to achieve a real time transmission both bricks need to have the same schedule (in order to open/close the time windows at the same time and don’t lose data), and they also need the PTP protocol to adjust their clock as similar as possible to each other.

The distribution creates a total reservation for priority queues 6 and 7 during the 4^{th} slot with the objective to secure the 6^{th} queue traffic. The 7^{th} queue in the MTSN Kit configuration is meant to transport only PTP traffic, that is of top

priority and sends little amounts of information. This is the main reason behind the 7th queue being allowed on all time slots.

Slot0:
0x000000BF-->0b10111111
Slot1:
0x000000BF-->0b10111111
Slot2:
0x000000BF-->0b10111111
Slot 3:
0x000000C0-->0b11000000

Slot/queue	0	1	2	3	4	5	6	7
0	1	1	1	1	1	1	0	1
1	1	1	1	1	1	1	0	1
2	1	1	1	1	1	1	0	1
3	0	0	0	0	0	0	1	1

There is an alternative way to configure the Time Aware Scheduler, introduced as an update to the MTSN Bricks. This new way presents a graphical interface that makes unnecessary the bit translation explained before. It also enables a new option: creating a number of time slots between 1 and 32.

The next figures show how to configure the time aware shaper, and how the interface shows the configured configuration.

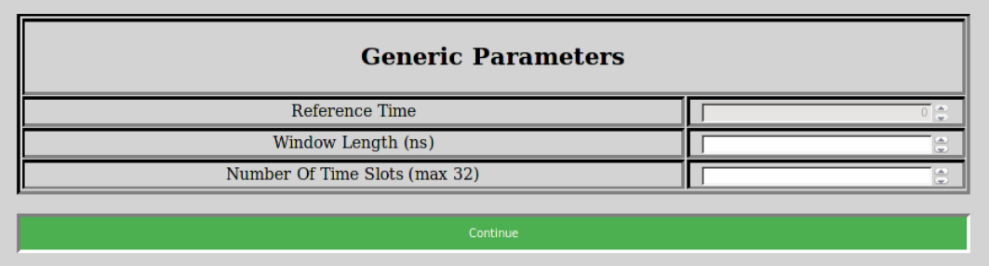


Figure A5.7 Time aware shaper graphic configuration

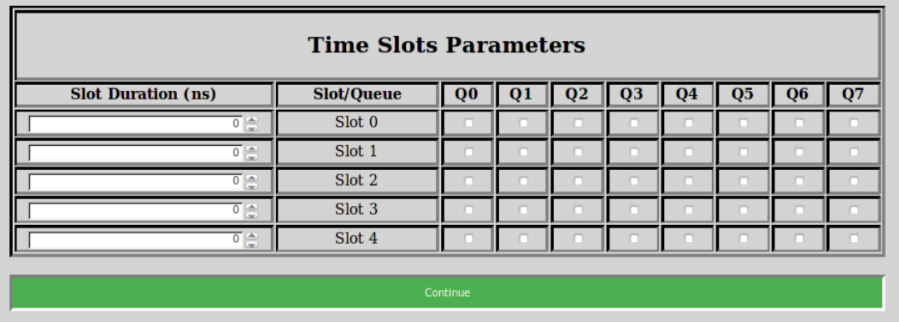


Figure A5.8 Configuration of PCP priorities usage from each time slot

Ip Address: 192.168.4.64

Port	0
Cycle Time	10000

Slot Duration	Slot/Queue	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
5000	Slot 0	1	1	0	0	0	0	0	0
5000	Slot 1	1	1	0	0	0	0	0	0

Port	1
Cycle Time	10000

Slot Duration	Slot/Queue	Q0	Q1	Q2	Q3	Q4	Q5	Q6	Q7
5000	Slot 0	1	1	0	0	0	0	0	0
5000	Slot 1	1	1	0	0	0	0	0	0

Figure A5.9 Time aware schedule configured in a MTSN Kit

Credit-Based Shaper

The CBS allows the definition of the maximum fraction of the port transmit rate that is available to a determined queue (BandwidthFraction). This fraction is determined by the configurable parameter PositiveSlope.

To configure the CBS there are a few parameters that must been take into consideration, the next list shows the parameters and their values in the MTSN configuration web tool during our experiment based on the featured demos.

- TSN Priority buffer : 6
- Queue Maximum Credit: 2436
- Queue Positive Slope: 2
- Queue Negative Slope:8
-

Credit Based Shaper

TSN Priority Buffer	7
Queue Maximum Credit	12176
Queue Positive Slope	10
Queue Negative Slope	0

Figure A5.10 Credit-Based Shaper configuration

The values are easy to understand knowing how the Credit-Based Shaper works. First of all the TSN Priority Buffer 6 relates to the VLAN PCP priority 6, the one that has been configured first.

Maximum Credit is calculated as it follows:
Maximum Credit = Max Interference Size*(Positive Slope/ Port Transmit Rate).
Port transmit rate is defined as 10 in this software implementation, negative and positive slope go from 0 to 10. So the division represents the percentage of usage of the port transmit capacity. The value 2 of “Queue Positive Slope” means that the selected priority has assigned a 20% of the bandwidth. Maximum Credit= Max Interference Size*(2/10). Maximum credit determines how much traffic a priority queue can send, when it becomes negative the priority queue needs to stop transmitting. During the time another queue is transmitting, the credit is increasing, when it becomes positive again the queue can transmit again. Max Interference Size= 12180. This value is assumed to be equal to the maximum frame size.

This experiment also features the priority 2 buffer as the counterpart to the 6 priority buffer:

- TSN Priority buffer : 2
- Queue Maximum Credit: 9740
- Queue Positive Slope: 8
- Queue Negative Slope:2

The values from Queue Positive Slope and Negative Slope are the opposite compared to the TSN Priority buffer 6, and the summatory of both buffer’s Maximum Credit is 12176, that is the value associated to the other TSN priority buffers. The explanation is that both buffers 2 and 6 are sharing the port transmit capacity. Calculating the Max Interference Size this time is: 12175, that makes sense because both buffers are sending packets of roughly the same size.

Finally we can observe the other buffers, all of them configured to transmit at 100% port capacity.

TSN Priority buffer: 0, 1, 3, 4, 5, 7
Queue Maximum Credit: 12176
Queue Positive Slope: 10
Queue Negative Slope: 0

Annex 6: Scheduling mechanisms

This Annex describes scheduling mechanisms useful to calculate schedules that can satisfy both the Ethernet and specific TSN constraints. Most of this material comes from [16].

Traffic model enhanced with TSN

On any Ethernet network, a full-duplex physical link between two nodes is characterized by the tuple { [va, vb].s, [va, vb].d, [va, vb].mt } [16], where va,vb are the two nodes at the ends of the link, s is the speed of the link, d is the propagation delay and mt is the macrotick of the link. As a side note, the macrotick represents an entire number of microticks, that are the ticks from an oscillator like a clock.

For TSN networks, the previous definition needs to be extended [16] with [va,vb].c, that represents the number of available schedule queues in the device, with a maximum value of 8. A periodic data transmission using one or multiple links is defined as a stream, and is expressed as: si = [[va, v1], [v1, v2], . . . , [vn−1, vn], [vn, vb]], where each element represents the route of the stream trough the network. A stream is defined through the tuple: {si .e2e, si .L, si .Ti}, where e2e is the maximum allowed end to end latency, L is the data size in bytes and T is the period. For TSN networks, another variable is needed: (si^ [va,vb]) .p, that represents the assigned queue for the instance of a stream on a specific device.

Scheduling constraints

To compute a transmission schedule, the scheduling algorithm needs a set of constraints, that represent both the network limitations and the desired transmission properties. This sections defines these constraints, both the general constrains that are present on any Ethernet network and the specific constraints associated with TSN protocols.

Basic Ethernet Constraints:

- **Frame constraint**: A frame belonging to a critical stream has to be scheduled between time 0 and its period given the periodic repetition pattern of critical traffic.

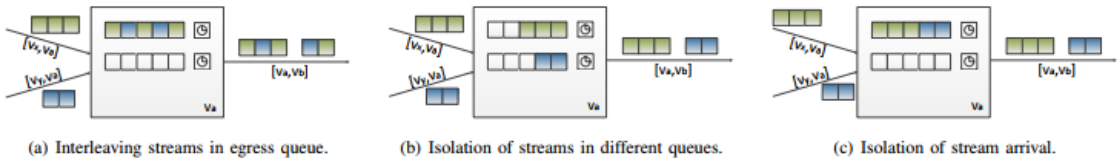
- **Link constraint**: Since there can only be one frame at a time on a physical link, no two frames that are routed through the same egress port may overlap in time.

- **Stream transmission constraint**: In order to ensure a low latency, the propagation of frames of a stream must follow a sequential order along the routed path. One important detail is the network precision, the worst case difference between the local clocks of two synchronized devise. The synchronization frames in the 802.1AS protocol have a lower priority than critical traffic to avoid introducing a delay. The constraint imposes that a frame can only be scheduled on a subsequent link after the complete reception on the previous link.

- **End-to-end constraint**: The difference between the reception of a stream at the destination and the transmission of the stream from the sender has to be less than or equal to the specified duration.

802.1Qbv Constraints:

802.1Qbv operates opening and closing timed gates on the queues of an egress port, not the order of frames in the queue. In Figure A6.1a, due to several factors, the arrival of frames is altered; therefore their order in the scheduled queue may be non-deterministic. Figure A6.1b presents a solution: not allowing streams arriving during interfering intervals to be placed in the same queue.



(a) Interleaving streams in egress queue. (b) Isolation of streams in different queues. (c) Isolation of stream arrival.

Figure A6.1 (a to c) Different methods to send frames on the egress queue (taken from [16])

On the other hand, figure A6.1c proposes allowing them to be placed in the same queue but ensure that the intended order and transmission time for all frames of the streams are preserved. Thanks to this example, a few constrains based on 802.1Qbv can be formulated:

- **Stream isolation constraint**: In order to isolate streams that are placed in the same queue we must ensure that they are isolated in the time domain on arrival. The constraint ensures that once a stream has arrived at the receiving device, no other stream can arrive at the same egress port until the first stream has been completely sent on the output port. This constraint is restrictive since an input where a high-rate stream has a period which is equal to or less than the combined transmission duration of all frames of a low-rate stream will not be schedulable with the stream isolation constraint (if there is only one queue per port).

- **Frame isolation constraint**: In order to prevent the restrictions created by the stream isolation constraint, this constraint allows frames interleaving between streams in a queue while in the same time

guaranteeing that the order on the output port is deterministic, enforcing that there are only frames of one stream in the queue at a time.

SMT-BASED schedule synthesis

Using heuristics (algorithm that only ensures a good solution or a good execution time) to create a scheduling algorithm has the disadvantage that do not search the entire solution space. To obtain an optimal algorithm, one idea is using Satisfiability Modulo Theories (SMT) [47]. SMT are designed to determine the satisfiability of first-order logical formulas against certain background theories like linear integer arithmetic. The aim of this scheduling algorithm for TSN networks is to find values for all individual frame offsets and queue assignments in each respective egress port of streams routed along the network such that the set of constraints are met, the frame offsets represent the open and close events for the timed-gate of the assigned queue of the frame/stream. The drawback of using SMT solvers is that for large networks they may take an unrealistic amount of time to solve the scheduling problem.
Some solutions are being studied to solve this problem, like scheduling only one stream at a time, adding it to the SMT context and then repeating the process taking the context into account.

Some TSN specific optimizations can be formulated to allow fastest algorithm work. Non-critical traffic may benefit from a larger number of priority queues with regard to their quality of service characteristics, or creating an additional variable for each device representing the number of required scheduled queues for high-criticality streams for each egress port are two of them.

The No-wait packet scheduling for TSN

While the IEEE 802.1Qbv standards define mechanisms to handle scheduled traffic, it does ont define algorithms to compute fine-grained schedules for the streams that need the mechanism of guard bands to isolate traffic with different priorities. The no-wait packet scheduling [17] is an adaptation of the no-wait job-shop scheduling (a constrained optimization problem that tries to schedule each operation on the corresponding machine such that no more than one operation is processed at the same time on any machine) for calculating TSN schedules yielding minimum network delay for real-time flows and compact schedules.

As described before, the CNC discovers the network topology using LLDP and exploits this info, programing the gate-drivers in the switches based on the computed schedules, and also provides the source hosts of the time-sensitive flows with a timestamp relative to the start of the program for injecting packets of scheduling traffic into the network. A time-sensitive flow then corresponds to a sequence of forwarding operations, one for each switch along the given path of the flow. Packets should be forwarded immediately without delay, which intuitively corresponds to the no-wait property of no-wait job-shop.

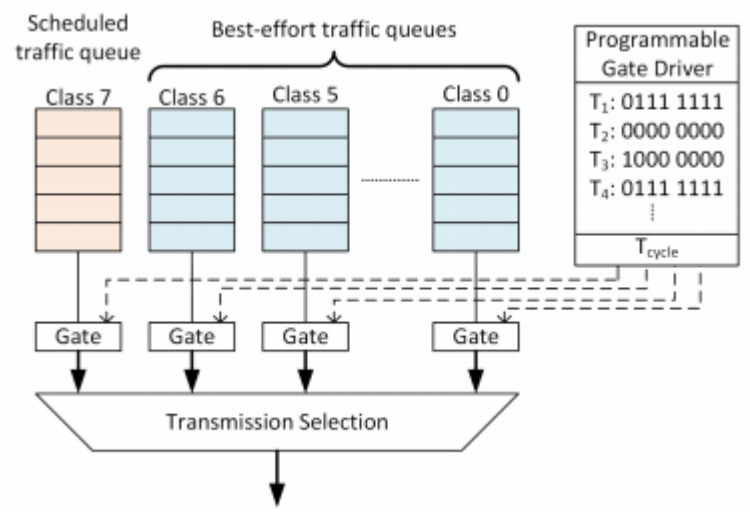


Figure A6.2 Time Aware Shaper configuration (taken from [17])

Typically, the TSN switches isolate scheduled traffic from best-effort traffic by means of so-called “guard bands” using the gating mechanism, assuring the transmission of an MTU-sized packed at the port.

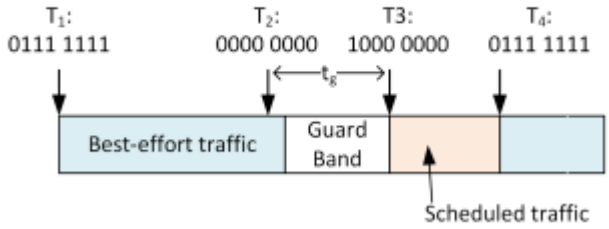


Figure A6.3 Guard band protecting scheduled traffic (taken from [17])

The algorithm’s goal is to define the times when packets are injected into the network by NICs at the source, and the times to open and close the gates for scheduled packets at switches, assuming a store-and-forward behavior for switches rather than cut-through forwarding. This assumption allows us to define sequential and ordered time intervals for propagation, processing, queuing and transmission.

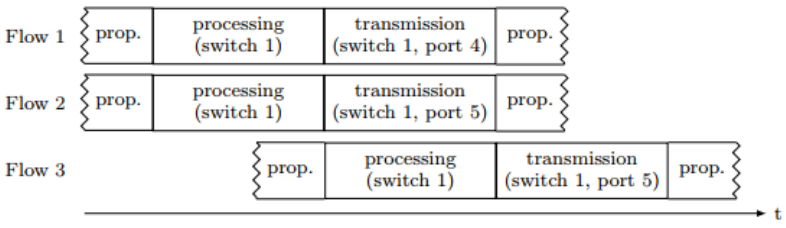


Figure A6.4 Time aware schedule (taken from [17])

In order to improve scalability, the algorithm presents heuristics for efficiently solving bigger networks, based on the Tabu [48] search algorithm. The resulting schedule is then subjected to a specifically designed procedure to reduce the number of gate-opening events for scheduled traffic, thus reducing the number of guard bands and conserving bandwidth for best-effort traffic.

The No-wait packet scheduling problem can be split into a time-tabling problem and a sequencing problem. The timetabling problem deals with computation of start times for all flows belonging to a totally ordered set of flows. The sequencing problem, on the other hand, deals with totally ordering the set of flows being scheduled such that the given time-tabling algorithm results in a schedule with minimal flow span. To reduce the number of gate opening events in the TSN schedule, the schedule needs to be compressed. Schedule compression is based on the principle that start times for certain operations may be delayed such that they end just before the succeeding operations begin on the corresponding machines. In terms of the schedules, it means that the transmission of a scheduled packet may be delayed to a time such that it is finished just before the next scheduled packet for transmission on the same port is available.

The No-wait packet scheduling problem using Tabu search algorithm calculates near optimal solutions for No-wait packet scheduling problem with respect to minimizing the flow span. Its execution times depends on the number of flows being scheduled. Overall, this approach scales to scheduling over 1500 time-sensitive flows. The algorithm for schedule compression results on an average reduction of 24 % for gate-opening events/guard bands.

Annex 7: OPC UA and TSN

OPC Unified Architecture (OPC UA) is a machine-to-machine communication protocol for industrial automation developed by the OPC Foundation [2]. Its main goal is to integrate the functionality of older OT communication standards, mostly the OPC family standards. OPC works using the master-slave paradigm, allowing for a universal standard to communicate the OPC server, designed to provide a common bridge for Windows-based software applications and process control hardware, with any vendor's OPC-based client. This way it can define a common interface that is written once and then reused by any business, SCADA, HMI or other industrial control equipment. OPC UA APIs are available in several programming languages: C, C++, Java, and .NET, Javascript (NodeJS) and Python.

- OPC UA distinguishing characteristics are:
- Focus on communicating with industrial equipment and systems for data collection and control.
- Open - freely available and implementable without restrictions or fees.
- Cross-platform - not tied to one operating system or programming language.
- Follows a Service-oriented architecture (SOA).
- Robust security.
- Follows an integral information model, which is the foundation of the infrastructure necessary for information integration where vendors and organizations can model their complex data into an OPC UA namespace take advantage of the rich service-oriented architecture of OPC UA. There were over 35 collaborations with the OPC Foundation currently. Key industries include pharmaceutical, oil and gas, building automation, industrial robotics, security, manufacturing and process control.

From a technical standpoint, it is possible to include real-time capacities to OPC UA [2], but it would imply an effort that will be better focused on implementing TSN capabilities to the technology, as explained in [3]. Some prototypes to the integration were presented at Hannover's Fair 2016 [3], and even if as today there is no final solution the overall industry opinion is that the efforts are a necessity for the evolution of industrial control protocols. The different approaches that some companies have created as vendor-specific solutions are called the "Industrial Ethernet", but all of these solutions were created without the cooperation of the IEEE, so there's no standard defined yet for an all-vendor solution. OPC UA TSN adds deterministic behavior to industrial Ethernet

networks, but also a number of new networking features, including time synchronization, ingress policing, seamless redundancy, frame preemption, scheduled traffic and stream reservation.

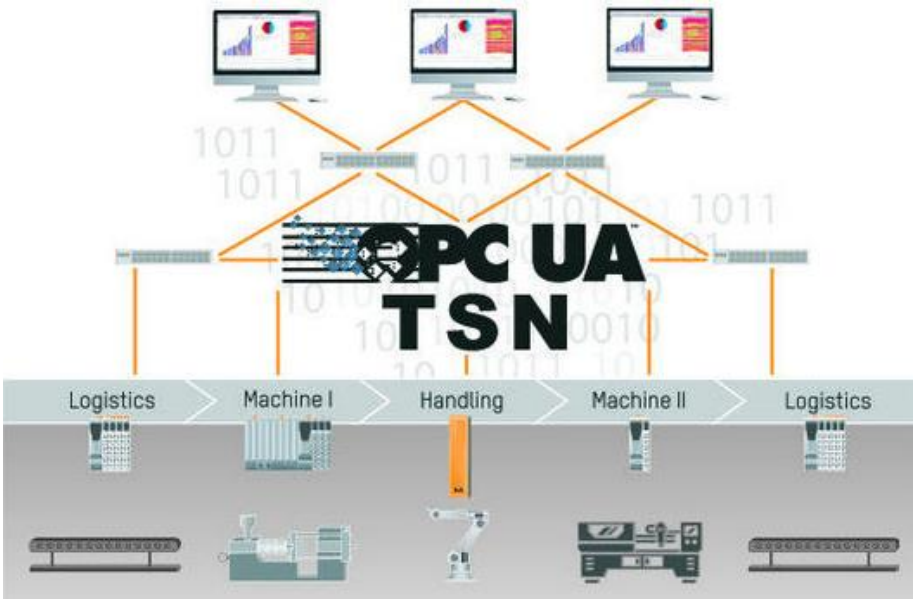


Figure A7.1 OPC UA TSN network (taken from [37])

TTTech [4], an Austrian based company, is currently selling their TSN-OPC UA "Starter Kit", that includes TSN-enabled switches and OPC UA compliant microcomputers for deploying a non-configurable network that allows companies to test their OPC UA components in a TSN network and understand the benefits in security (preventing against DDOs or Denial of Service attacks) and achieving Guarantee of Service.