Universitat de les Illes Balears

Escola Politècnica

Memòria del Treball de Fi de Grau

Design, implementation and validation of a system to automatically configure the scheduling of real-time traffic in Time Sensitive Networking (TSN) switches

Antonio Jorda Aparicio

Grau d'Enginyeria Electrònica Industrial i Automàtica

Any acadèmic 2018-19

DNI de l'alumne:41522403C
Treball tutelat per Inés Álvarez Vadillo y Manuel Barranco
Departament de Ciències Matemàtiques i Informàtica

Paraules clau del treball:
TSN, real-time traffic

Universitat de les Illes Balears

escola politècnica superior

Final Degree Report

GRAU D'ENGINYERIA ELECTRÒNICA INDUSTRIAL I AUTOMÀTICA

Design, implementation and validation of a system to automatically configure the scheduling of real-time traffic in Time Sensitive Networking (TSN) switches

ANTONIO JORDA APARICIO

Tutors

Inés Álvarez Manuel Barranco

Escola Politècnica Superior
Universitat de les Illes Balears
Palma, July 1, 2019

Contents

List of Figures

List of Tables

ABSTRACT

During the last years there is a growing interest in interconnecting and integrating systems that rely on *Information Technologies* (IT) with systems that rely on *Operational Technologies* (OT), i.e. on technologies used in industrial systems to monitor/control a plant. This is mainly because of the potential benefits of sharing information at multiple levels, from the physical world to decision-making systems. Some examples of these benefits are an increased manufacturing flexibility, increased maintainability, lower cots, etc.

In the context of the OT this interconnection/integration has been coined as the *Industry 4.0*. Since the interoperability is the core of the Industry 4.0, the different levels of a system architecture have to be interconnected by means of several communication protocols that can be easily integrated, or by means of a single protocol that satisfies the requirements of all levels.

An appealing protocol to be used in this sense is Ethernet; not only because of its low cost and huge bandwidth, but also because it is the de-facto link layer protocol in IT systems. In the last decades, Ethernet has been adapted to meet the communication requirements imposed by industrial control systems; specially to support real-time communications. Unfortunately, a myriad of industrial-Ethernet protocols exist, which often cannot be easily integrated among them.

In order to cope with this problem and to make Ethernet an actual enabler for the integration of IT and OT systems, the *Institute of Electrical and Electronics Engineers* (IEEE) is developing a set of standards known as *Time Sensitive Networking* (TSN). The main contributions of TSN standards consists of providing mechanisms for time synchronization, reliability, network management, and deterministic and bounded communication latency.

Deterministic and bounded communication latency is fundamental to provide real-time communication. The main mechanism TSN standards propose in this regard is the *Time Aware Shaper* (TAS). The TAS is a set of traffic shapers, each located in a different switch and end node of a TSN network, that adequately forward/transmit the frames to guarantee the *Quality of Service* (QoS) and the real-time requirements of different classes of traffic.

To correctly operate, the TAS needs to be configured with the appropriate traffic scheduling information. Unfortunately, so far, the TAS is manually configured in each switch and node, which means that the value of each scheduling parameter has to be set one by one. This does not only constitute a tedious and error-prone process, but it also makes it impossible to automatically reconfigure the scheduling; which is a feature that TSN (and the companies that are pushing this technology forward) wants to support in the near future as a means to provide flexibility.

The objective of this project is to design, implement and validate a system that allows automatically configuring the TAS's scheduling parameters of the switches of a TSN network.

CHAPTER 1

INTRODUCTION

1.1 Background and motivation

An embedded system is a computational system (hardware and software) aimed at controlling another system, normally a physical one, e.g. an embedded system that controls the stability of a road vehicle. Embedded systems are widely used in various fields such as factory automation, automotive industry, telecommunications and satellite systems among others.

An embedded system forms part of the system it controls, and thus it is normally restricted [7] in terms of weight, size, energy consumption, and hardware/software resources. In addition, an embedded system sometimes has to operate satisfying the time constraints imposed by the system under control. When this happens, the embedded system is said to be a real-time (RT) system, i.e. a system that has to carry out its functionality within specific time limits.

When an embedded system is constituted by a set of nodes that cooperate and that communicate through a network, it is referred to as a *Distributed Embedded System* (DES). Communication protocols for DESs have been traditionally designed to cope with the requirements of specific fields. As a result, in the context of the *Operational Technologies* (OT), there are a myriad of different protocols, which then are difficult to integrate with each other.

This limitation on easily integrating different protocols is an important issue to be overcome. In particular, note that nowadays there is an increasing interest in integrating *Operational Technologies* (OT) and *Information Technologies* (IT) [8], so as to exploit the potential of ITs in fields such as factory automation. In fact, this trend has been recently materialised in the emergence of concepts such as the Internet of Things, Smart Grid, Smart Cities, Vertical Integration (between IT and OT), and Industry 4.0.

A communication technology that can act as an enabler/catalyst for integrating OT and IT is Ethernet. On the one hand, Ethernet is the de facto standard data link layer protocol in the IT context. On the other hand, given its classical advantages, e.g. low cost, high bandwidth, wide know-how, etc., Ethernet is a communication

technology that has been adopted in the context of OT for many years now. Moreover, since networks nowadays tend to be more and more larger and complex, there is a growing interest in exploiting other Ethernet advantages in the IT/OT context, e.g. Ethernet-based technologies like Software-Defined Networking (SDN), Ethernet native support for TCP/IP networks, etc.

However, it is important to note that Ethernet presents an important limitation that may discourage its use in the OT context. Specifically, the collision-based Medium Access Mechanism (MAC) of Ethernet does not provide a deterministic and bounded medium access delay/latency, which are vital for guaranteeing real-time communications. Many industrial Ethernet-based protocols have been developed in the context of OT so as to guarantee real-time communications. This has result in flooding the market with numerous Ethernet-based industrial protocols that are not totally compatible among them and therefore require gateways to interoperate.

In order to take profit from the above-mentioned Ethernet advantages in the automation domain while paving the way for integrating OT and IT systems, the IEEE 802.1 TSN Task Group is developing a set of Ethernet standards under the name of *Time Sensitive Networking Standards* (TSNS or TSN for short).

Roughly speaking a TSN network consists of a set of interconnected full-duplex (TSN) Ethernet switches (forming a multi-hop topology); and a set of (computational) TSN nodes (each connected to one or more TSN switches). The switches and the nodes are equipped with mechanisms that provide tight time synchronization, reliability (fault tolerance), traffic management, bounded communication latency for real-time communication, etc.

In order to guarantee real-time communication, TSN defines and guarantees the *Quality of Service* (QoS) of different classes of traffic. The definition of these classes depends on the specific TSN standard being considered. However, generally speaking, one can say that TSN distinguishes 3 classes of traffic, namely: *Time Aware traffic Class*, which corresponds to hard real-time traffic; *Class A traffic*, which is devoted to soft real-time traffic; and *Class B traffic*, which carries best-effort (non real-time) traffic. Another important aspect of TSN is the concept of *stream*. An stream can be understood as a flow of information that a source node/switch (known as *talker*) transmits to one or more destination nodes/switches (*listeners*).

One of the core TSN standards for real-time communication is the IEEE 802.1Qbv. This standard specifies what is known as the *Time Aware Shaper* (TAS). Basically, the TAS consists of a set of traffic shapers - each located in a given TSN switch/node - that adequately forward/transmit the frames so as to guarantee the QoS of the different traffic classes in general, and of the Time Aware traffic Class in particular. More specifically, to guarantee the real-time requirements of the Time Aware traffic, the TAS organizes this traffic in a *Time-Triggered* (TT) fashion, i.e. each node and switch respectively transmits and forwards frames in specific time slots following a given *configuration*.

To better understand the concept of *configuration* please recall that, from the functional point of view, a network can be divided into the *control plane* and the *data plane*. The control plane is the level that takes the decisions about when/where to transmit/forward each frame. The data plane is the level that actually transmits/forwards the frames following the decisions of the control plane. Thus, the *configuration* can be understood as the information (of the control plane level) that the nodes and the switches use to transmit/forward frames. For the specific case of the TAS, the configuration is the TAS's

scheduling information used to transmit/forward each frame through the adequate port at the adequate time slot.

The configuration itself is of utmost importance in the context of TSN, since it actually contains the information that switches and nodes follow to guarantee the QoS and/or the real-time requirements of the different streams. Unfortunately, although the IEEE 802.1Qbv already specifies most of the TAS mechanisms, it does not specify how to set a given configuration up on the switches and the nodes.

This lack of specification is slowing down the development of TSN equipments and networks. In this sense note that, although TSN is still under development, important companies such as TTTech [9] have already started to produce and sell TSN equipment. However, as far as we know, no company that produces TSN equipment provides any system to automatically configure the devices of a TSN network. Instead, every vendor offering TSN products makes it necessary to use a vendor-specific software to configure the devices manually, i.e. the network administrator (a human) has to setup, one by one, the value of each one of the parameters that form part of a given configuration. Obviously, this kind of configuration procedure is tedious and error-prone. It could be enough to manually test small-sized networks; but it is not adequate when tests need to be automated, when tests need to be exhaustive, or when the size or the complexity of the TSN network increases.

The objective of this project is thus to design, implement and validate a system that allows to automatically set the TAS's configuration (scheduling information) on the switches of a TSN network. The automatic configuration of the TSN nodes is postponed for future projects.

1.2 Project objectives

As stated above, the general objective of this Bachelor Thesis is *to design, implement and validate a system that allows to automatically set the TAS configuration on the switches of a TSN network*. We will refer this system to as the *automatic configuration system* from hereon. At this point and before continuing, let us to briefly outline what actions the automatic system should carry out.

First, the automatic system must accept a reading file containing the value for each parameter that forms part of the TAS configuration. Note that the content of this file reflects the scheduling information each TSN switch must use. This information can be very complex to calculate and, thus, in a real system it would be generated by an appropriate scheduler (typically by means of a *cyclic executive* scheduler [10]). Such a scheduler is out of the scope of this project and, thus, we will generate the reading file manually for validation purposes. Once the system is fed with the reading file, it must automatically translate this file into a *data model*, i.e. a data structure. This data model must be understood by each entity that in, each TSN switch, is responsible for consulting/updating the database that stores the portion of the TAS that corresponds to that switch, i.e. the *TAS database*. Finally, the system has to communicate with said entities; so that each one of them can update the TAS database of its TSN switch. In this sense, on the one hand, the system must automatically generate and properly encode what we call a *configuration file* that the mentioned entities can parse and process to obtain the data model (and then update the TAS database accordingly). On

the other hand, the system has to dialog with these entities using an adequate *network management* protocol to transfer the configuration file.

Note that, analogously to what has just been discussed about the actions needed to configure the TAS, the system must allow to automatically retrieve all the TAS configuration from each TSN switch. In this way, the network administration, e.g. a human, can check that the configuration is correct.

It is important to clarify that this project does not include the design/implementation of any internal part of the TSN switches, including the entities responsible for consulting/updating the TAS databases. This is so because those parts are already provided within the TSN equipment in general, and within the TSN switches we use in this project in particular.

Taking into account the just-mentioned clarifications, the specific goals of this projects are as follows:

- To design an automatic configuration system of the TAS, respecting the TSN specifications (which as already said, do not include the description of the way in which the TAS must be configured).
- To implement the designed automatic configuration system.
- To validate the implementation of the designed automatic configuration system on a real prototype.

1.3 Completed tasks

In order to accomplish the objectives described in the previous section, we have divided the project into different phases. Later on, in Section 3, we will describe the methodology we have followed to carry out these phases, as well as a timeline that shows the time invested in each one of them.

In any case, to better understand the scope of this project, let us to briefly introduce each one of these phases next.

Study of the relevant TSN specifications

Prior to addressing the design of the automatic configuration system, I had to study the TSN specifications that are relevant to the scope of this project. First, I had to understand how the TAS works. Second, I needed to study the parts of the TSN standards [4] that specify different TSN network architectural options that can influence the design of the configuration system itself. Third, I had to learn the specification of the data model of the TAS, as well as the language needed to write that model. Specifically, as it will be explained later, TSN standards indicate that the language to be used for this purpose is the *Yet Another Next Generation* (YANG)[11] data modeling language. Fourth, I needed to study the different network managements protocols TSN standards consider as appropriate. Specifically, as it will be explained later, we chose the *Network Configuration Protocol* (NETCONF) [12].

Design of the automatic configuration system

In this phase I designed the architecture of the automatic configuration system, as well as the set of mechanisms provided within this architecture.

First, as just pointed out above, the architecture of the automatic configuration system has to be designed in such a way that it can be used with one of the network architectural options considered in the TSN standards (these options are known as *architecture models* in TSN). Specifically, as it will be explained in Section 4.2, I have designed the architecture of the automatic configuration system to adhere to the *Fully centralised model*[4].

Second, I had to design each one of the mechanisms provided within this architecture. Roughly speaking, this includes the design of the procedure (steps) needed to configure the TAS; the reading file of the automatic configuration system, i.e. the *configuration file*; the YANG data model of the TAS; the configuration file (which will be encoded in XML) that will be exchanged with the TSN switches via the network management protocol; and what we call the *User database*.

Concerning the last one of these mechanisms, i.e. the User database, note that it is used to store different configurations. Using the User database, the user does not need to have all the reading files but only those ones that have not been stored in the User database yet.

Search and selection of development tools

In this phase I had to consider different tools for implementing the architecture and the mechanisms of the automatic configuration system. In Section 5, we will explain the tools we finally chose to implement the system; with special emphasis in those we needed for supporting the network managament protocol we finally used in this project, i.e. NETCONF.

Implementation and testing of the mechanisms

In this phase I implemented all the mechanisms that are part of the architecture described in the design. The implementation of the mechanisms was carried out one by one, so before starting implementing the next mechanisms I realized the convenient tests to ensure that the mechanism is faultless.

Integration and validation

The last step in the project consisted in integrating the mechanisms in the switches and testing that the automatic configuration system works as it is expected. As it is described in the chapter 7, we have carried out tests first simulating a switch on the PC and then on the real switches.

1.4 Document structure

This document has seven chapters excluding the introduction. Chapter 2 describes the technologies that have been used during the execution of the project.Concerning this chapter, the most relevant sections are the ones that describe the IEEE 802.1Qbv and the protocol NETCONF. Chapter 3 illustrates the different phases in which the project has been divided and compares them with the phases of the waterfall model.

Chapter 4 explains the provided equipment and how has been done the design that later has been implemented. Chapter 5 contains the tools that have been selected as a platform for the implementation that is explained in Chapter 6. Chapter 7 contains

how the mechanisms have been integrated in the switches and the tests that we ran for ensuring the correct performance of the mechanisms.

Chapter 8 is a summary of the most relevant parts of the project and also includes suggestions for future work based on what has been accomplished.

Finally Appendix A contains all the tests that have been carried out in Chapter7.

CHAPTER 2

STUDY OF THE RELEVANT TSN SPECIFICATIONS

This chapter briefly explains the concepts that are fundamental to understand this project. As important parts, it describes TSN and some of its standards. Notice that the IEEE 802.1Qav and the IEEE Std 802.1Qcc are particularly important for the development of this project.

2.1 TSN

2.1.1 Ethernet

Back in 1973, Robert Metcalfe and David Boggs, members of the Xerox Palo Alto Research Center (PARC), were assigned to create a local-area network (LAN) to connect the first personal workstation with a graphical user interface to a laser printer. The result of their work was Ethernet, the first high-speed LAN that allows to connect numerous computers at the same time through a single shared cable called Ether.

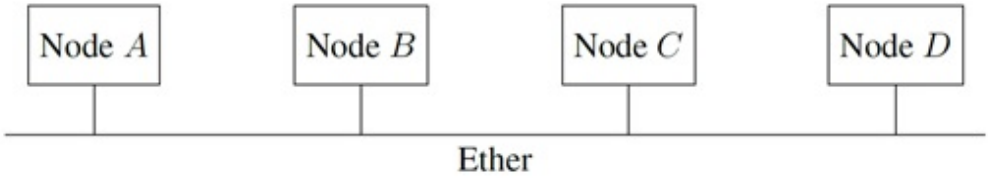


Figure 2.1: First Etherneet topology [1]

In order to create their network, Metcalfe and Boggs, looked at the wireless computer network ALOHAnet which was used for the communication between the Hawaiian Islands. The ALOHAnet protocol works in the following way, first a station transmits a message through the shared medium whenever it wants. Then it waits for an acknowl-

edgment message. If the acknowledgment is received before a certain time—known as the timeout—it means that the transmission succeeded and if it is not received the station assumes that a collision occurred due to another station transmitted simultaneously. When a collision is detected, both transmitting stations wait a random amount of time —called a random backoff time— and then retransmit the message. This way, if the stations have chosen different times, the corruption will not occur. Otherwise, a collision occurs again, and it is solved in the same manner but increasing the possible values of the backoff time and reducing the probability of a collision. Notice that as the traffic increases, the collision probability increases as well. Based on this idea, Metcalfe developed a protocol to detect collisions called *Carrie Sense Multiple Access with Collision Detection (CSMA/CD)*. Besides, he added a source and destination address in each message and a checksum (CRC).

In CSMA/CD, each station before transmitting listen for other transmissions in the channel, if there is not any then it transmits. This reduces the probability of collision compared with the ALOHAnet protocol. However, this does not extinguishes the probability of a collision. For example, several stations could see that there is not anyone sending data through the channel and decide to transmit at the same time. The other element of CSMA/CD is the collision detection that allows the stations to stop the transmission of the frame if it is corrupted. While a station is sending data it also monitors its own transmission and if it detects a collision stops. This is possible due to Ethernet uses a coaxial cable as the shared medium instead of the air as ALOHAnet. As we have seen, due to the backoff time the classic Ethernet is not deterministic and therefore not suitable for time-real applications. Notice that the collisions could be avoided with a full-duplex connection but there is still the problem of managing the frames in order to ensure that the information arrives on time.

Firstly, Ethernet was an in-house system but in 1976 Metcalfe and Boggs published a scientific paper, *Ethernet: Distributed Packet-Switching for Local Computer Networks*. Few years later, in 1980, the first Ethernet standard was published by DEC, Intel and Xerox. This standard became known as the DIX Ethernet standard. At the same time the Institute of Electrical and Electronics Engineers (IEEE) began to develop a new set of standards in the IEEE 802 committee. The first Ethernet standard published by the IEEE was the 802.3 in 1983 and was approved in 1985. Even though the 802.3 was published with the name *Carrier Sense Multiple Access with Collision Detection (CSMA/CD)*, it is mostly known as *Ethernet*.

2.1.2 Motivation of TSN

Since the first standard was released, Ethernet expanded its domain. By the end of the 20th it was the last LAN standing and the only choice for office networks. Then, it kept gaining place in other fields including the embedded systems where it is already being used but not as the major one. The fieldbuses remain as the main option for such purpose. In an industrial environment appears specific requirements: a cable that resists hazardous environments as extreme temperatures, mechanical vibration, electronic magnetic interference (EMI), etc and determinism. As it has been explained before, Ethernet uses CSMA/CD and every time a collision occurs the station waits a backoff time before retransmitting again but there is not any guarantee that a collision will not occur again. For this reason, classic Ethernet is not a bounded and a determin-

istic protocol and not suitable for industrial applications . In order to make Ethernet to satisfy the requirements for each industry, several specific protocols have appeared. Some of them are PROFINET, POWERLINK, EtherCAT and EtherNet/IP.

During the past few years it has increased the interest for interconnecting all the information. Followings this, it appeared the Industry 4.0 which aim is to use Internet for create smart factories. The idea is to introduce customization and flexibility to the mass production technologies. To accomplish this purpose, machines have to be able to collect data, analyse it and respond depending on that data. The core of this new industry is the *Cyber Physical Systems* (CPS). CPSs are the combination of computational entities, networking and physical processes whose purpose is to control a physical process adapting itself to the environmental conditions in real time.

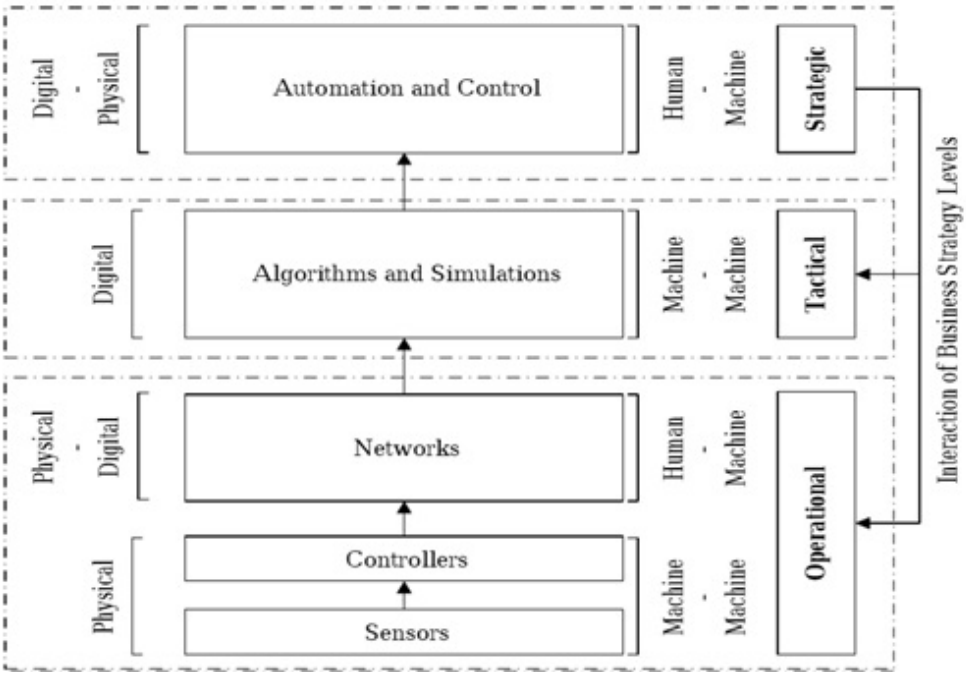


Figure 2.2: Elements of Cyber Physical Systems [2]

As it is seen, Industry 4.0 is an architecture that demands a high level of interoperability and flexibility, but this is very difficult to achieve because of multiple variants of protocols. In order to interconnect different protocols it is needed to implement protocol conversion gateways that are very expensive. Industry 4.0 needs a common communication protocol that connects the equipment from different vendors and satisfy the requirements of the industrial applications. Nowadays the candidate that fulfils all these conditions is the *Time-Sensitive Networking* (TSN). TSN is a set of standards that make Ethernet suitable for real-time applications and therefore appropriate for the Industry 4.0 and analogous concepts.

2.1.3 Relevant standards

Audio Video Bridging (AVB), was an initiative to transport audio and video streams over IEEE 802 networks. Basically its protocols provide time synchronization, bandwidth

allocation and traffic shaping. In 2012, AVB evolved to TSN expanding the scope of the standard. The working group IEEE 802.1 which is in charge of the project are designing the standards to make Ethernet deterministic to meet the different requirements of the industry. TSN is a technology that works on the data link layer of the OSI model [13].

Component	Audio Video Bridging (AVB)	Time Sensitive Networking (TSN)
Time synchronization	IEEE Std 802.1AS	IEEE Std 802.1AS P802.1AS-Rev
Reliability		IEEE 802.1Qca IEEE 802.1Qci IEEE Std 802.1CB P802.1CBdb P802.1AS-Rev
Resource management	IEEE 802.1Qat	IEEE 802.1Qat IEE Std 802.1Qcc P802.1Qcp P802.1Qcw P802.1ABcu P802.1CBcv P802.1CS
Latency	IEEE 802.1Qav	IEEE 802.1Qav IEEE 802.1Qbu IEEE 802.1Qbv IEEE 802.1Qch P802.1Qcr

Table 2.1: AVB and TSN components

The table 2.1 specifies the main standards of AVB and TSN respectively. It is important to know that the documents starting with P are not finished, but under development. To describe some of the standards depicted in the table, first we need to introduce some definitions:

1. TSN stream. Describes the time-critical communication between end stations. Each flow has its own time requirements that must be satisfied by the devices that are involved in the flow.

2. End stations. These are the source and destinations of the TSN flows, they are also known as Talker and Listener respectively.

3. Bridges. Special bridges for transmitting the Ethernet frames of a TSN flow, from Talkers to Listeners, according to the schedule.

The most relevant standard regarding to time synchronization is the IEEE Std 802.1AS. It is based on *Precision Time Protocol* (PTP), therefore the IEEE Std 802.1AS describes a master-slave protocol to synchronize real-time clocks in the stations. Basically, PTP provides the same time for each station. IEEE Std 802.1AS defines a network master—known as Grandmaster—that sends the information to the stations.

The revision of the standard—P802.1AS-Rev—introduces new features required for time-sensitive applications. The main contribution is the spatial redundancy for the synchronisation Grandmaster.

Reliability is another of the important aspects of TSN. As a result, there are several standards concerned to provide it. Among them, we can highlight the following:

1. The amendment IEEE 802.1Qca —Path control and reservation— will provide explicit path control, bandwidth and stream reservation and redundancy for data flows between two stations. There are to possible modes; in the first mode the network manager can define the paths and deploy the logical topology; in the second mode the redundant paths are calculated autonomously by the network components without depending on the network manager.

2. The IEEE 802.1Qci—Per-Stream Filtering and Policing amendment— specifies mechanisms to perform frame counting, filtering, policing and service class selection for a frame based on the particular data stream to which the frame belongs.

3. The standard IEEE Std 802.1CB defines the mechanisms, for bridges and end stations, to identify and replicate frames for redundant transmission and to identify duplicate frames and eliminate them.

The resource management is in charge of describing the mechanisms and services to control the configuration of the network. Hereunder are described some of the standards concerned to the resource management. The main contributions to the resource management are:

1. IEEE 802.1Qat is the standard that defines the Stream Reservation Protocol (SRP). The protocol allows the stations to register their willingness to send or receive to/from specific streams, and it spreads the information through the network. SRP has been implemented on top of the protocol Multiple Registration Protocol (MRP) In TSN the streams are classified in two categories, Class A and Class B. Class A is for streams with tighter latency requirements whereas the Class B is for best-effort streams.

2. IEEE Std 802.1Qcc is and amendment to improve the already existing Stream Reservation Protocol (SRP) but it also introduces new architectures. Since it is a relevant standard for this project, it is explained in detail in a subsequent section.

3. P802.1Qcp, P802.1Qcw, P802.1ABcu and P802.1CBcv are different amendments that define YANG models for different purposes. For example, the P802.1Qcw is in charge of defining YANG model for Scheduled Traffic(Qbv),Frame Preemption (Qbu) and Per-Stream Filtering and Policing (Qci). Simply speaking, YANG is a data modelling language. The wide explanation of YANG is in the section 2.3.

Finally, there is a group of standards whose purpose is to achieve bounded low latency, therefore real-time transmission.

1. In the 802.1Qav amendment, the Credit-based shaper (CBS) is standardised. It ensures soft real-time for the classes A and B, defined previously in the 802.1Qat. The CBS algorithm defines credits associated to each of the classes. The frames from a class can be transmitted whenever the credit of the class is positive and there are not other frames being transmitted (no conflicting frames). The credit is increased, at a rate of *idleSlope*, when the frames are waiting to be transmitted, whereas it is decreased, at a rate of *sendSlope*, when the frames are transmitted. To bound the maximum and the minimum number of credits that can be accumulated the parameters *hiCredit* and *loCredit* are defined.

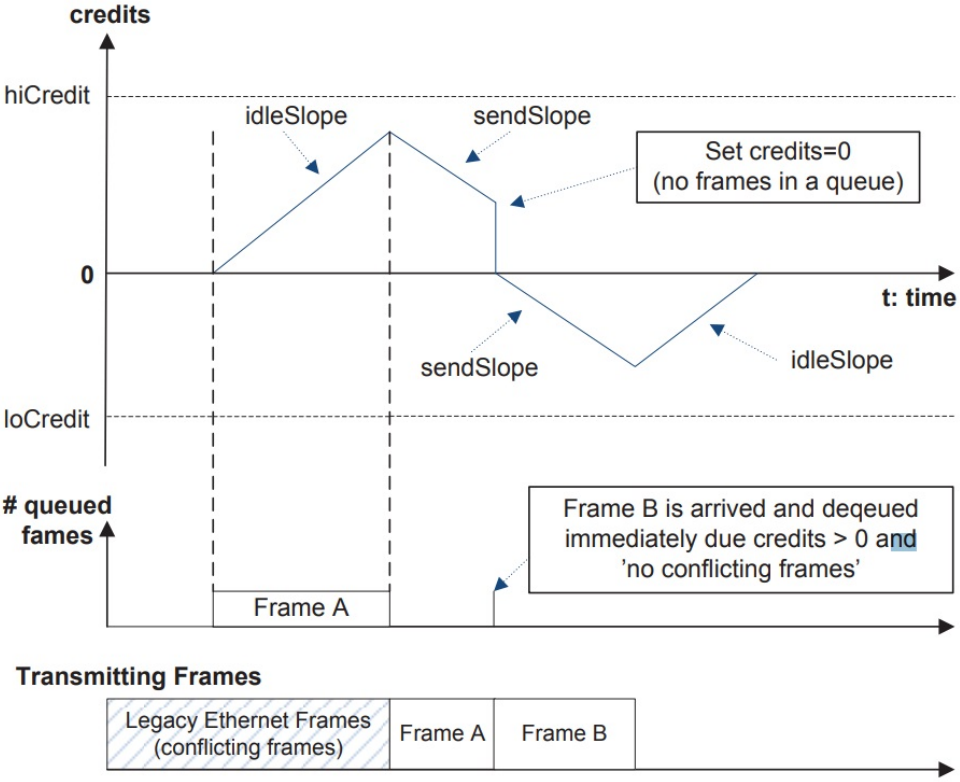


Figure 2.3: Credit-based shaper (CBS) operation [3]

2. The IEEE 802.1Qbv is the standard for the Time Aware Shaper (TAS) and defines the process to send scheduled traffic. Since it is a fundamental part of the project there is a complete description of the TAS in a subsequent section.

2.1.4 IEEE Std 802.1Qcc (Stream Reservation Enhancements and Performance Improvements)

IEEE Std 802.1Qcc—Stream Reservation Enhancements and Performance Improvements—is an amendment that provides enhancements for the configuration of TSN networks. These enhancements are achieved by improving the Stream Reservation Protocol, describing new architectures and specifying a software interface between the user and

the network. The main changes in the SRP are the creation of more traffic classes including the possibility to define new classes by the user and the *Time Aware traffic Class* that is hard-real time traffic and making SRP deterministic with bounded time for reservations. The software interface that allows exchanging information between users, that can be Talkers or Listeners, and the network, it is called *User/Network Interface* (UNI). Once defined the UNI the standard describes three possible architecture model: the *Fully distributed model*, the *Centralised network/distributed user model* and the *Fully centralized model*.

The *Fully distributed model* defines a direct communication between the users and the network through the UNI. The user sends its requirements to the network and the requirements are propagated through the Bridges. The suggested protocol to be used as the UNI and to send the information between bridges is the SRP. This architecture is only suitable to configure the Credit-based shaper.

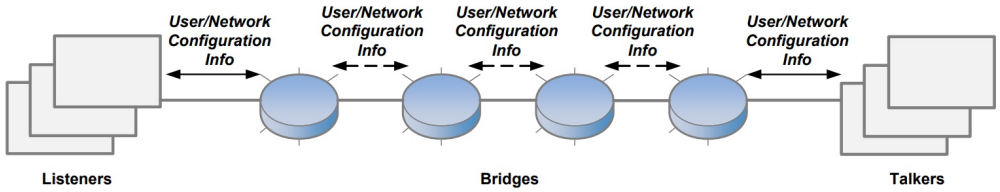


Figure 2.4: Fully distributed model [4]

In the *Centralised network/distributed user model* exists the *Centralized network Configuration* (CNC) entity. As in the *Fully distributed model*, the Talkers and Listeners send their requirements directly to the network (solid arrows), but the configuration information is sent from the CNC (dotted arrows). To perform the configuration the CNC uses a remote network management protocol that is depicted as dotted arrows in the figure 2.5. Notice that neither the Listeners nor the Talkers are necessary involved in the remote network management protocol. The purpose of the CNC is to perform computation operations, needed in some TSN features, in a single entity. For this reason, the CNC must know the physical topology of the network and the capabilities of each bridge. Moreover, the CNC configures the bridges connected to end stations, so those bridges directly send the information to the CNC (dashed arrows). The standard defines that the CNC can exist in an end station or in a bridge, but it does not give any guideline to make the decision giving totally freedom to the designer to choose one or the other depending on the specific characteristics of the application. Using this architecture, the following TSN features, among others, can be configured:

- Credit-based shaper (Qav)
- Frame preemption (Qbu
- Scheduled traffic (Qbv)
- Cyclic queuing and forwarding (Qch)

The UNI protocol suggested in the standard, for the *Centralised network/distributed user model*, is the SRP. Some options proposed from the standard for the remote network

management protocol[1] are Simple Network Management Protocol (SNMP), NETCONF (IETF RFC 6241) and RESTCONF (IETF RFC 8040).

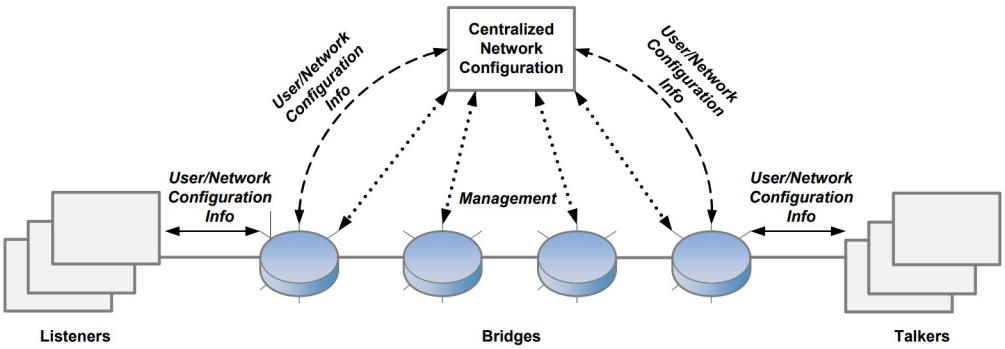


Figure 2.5: Centralised network/distributed user model [4]

The *Fully centralised model* is similar to the *Centralised network/distributed user model* with the difference that exists a *Centralised User Configuration* (CUC) entity over the CNC. In this architecture the UNI is situated between the CNC and the CUC and the requirements of the network are exchanged between the CUC and the CNC. In addition, the CUC obtains the requirements of the end stations through a protocol user-to-user, but this protocol is outside the scope of this standard. The *Fully centralised model* is suitable to configure the same TSN features as in the *Centralised network/distributed user model*:

- Credit-based shaper (Qav)
- Frame preemption (Qbu
- Scheduled traffic (Qbv)
- Cyclic queuing and forwarding (Qch)

[1] A network management protocol is a protocol that defines the processes and procedures to manage, monitor and maintain a network.

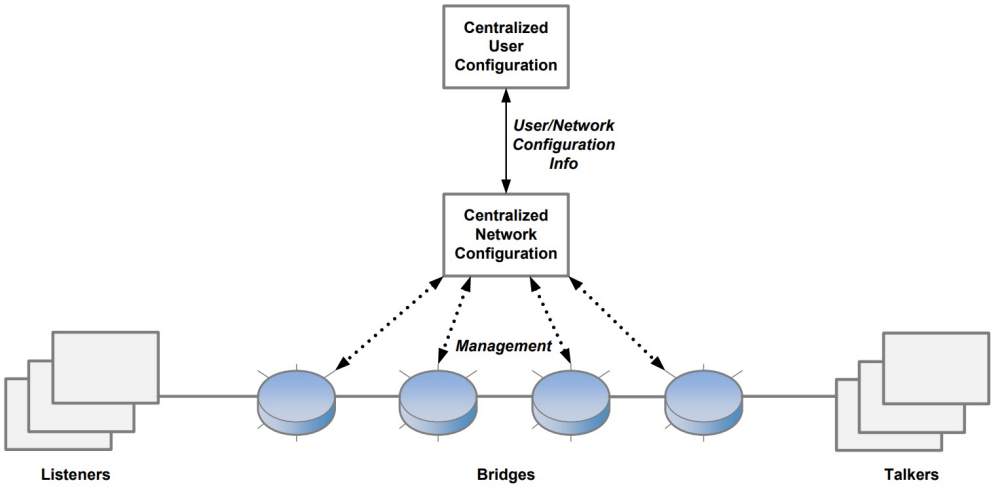


Figure 2.6: Fully centralised model[4]

2.1.5 IEEE 802.1Qbv (Time Aware Shaper)

The IEEE 802.1Qbv —standard for the Time Aware Shaper (TAS)— contributes to achieve a bounded latency, and it provides a mechanism to ensure that critical traffic is transmitted in a bounded time. The idea behind this mechanism is to divide the transmission, so that there is a time designated to transmit critical traffic and another for the soft traffic. Following this idea there is a protected window where only critical traffic can be transmitted. However, a protected window is not enough because if a soft traffic frame is transmitted just before the start of the protected window (T1) then it would interfere. To solve this problem a guard band is placed before the protected window. The duration of the guard band is the same as the transmission time for the maximum-sized frame (Figure 2.7 part A) and therefore the start of the guard band (T0) is fixed. If the length of the frames queued can be known by the implementation then the guard band can be variable (Figure 2.7 part B) so the start of it varies depending on the frames in the queues.

To impose the communication previously explained, the standard associates a gate to each queue that at the same time is associated to one traffic class (there are a total of 8 traffic classes defined).To control whether the traffic from one class have to be transmitted or not, there is a transmission gate for each queue. The transmission gate has two possible states, open or closed. If the transmission gate from a queue is open, then the frames are selected to be transmitted; but if the state is closed, then no frames are selected for the transmission. To control the state of the transmission gate there is a *gate control list* (also called *configuration* in this document) for each port. The gate control list contains a list with two parameters:

- *GatesState*: it indicates the state of each gate of the port
- *TimeInterval*: it is the quantity of time that will pass after the *GatesState* is set.

The procedure consists in setting the *GatesState* to the value indicated in the first *gate control list* element. After the *TimeInterval* has elapsed, the next *GatesState* is set

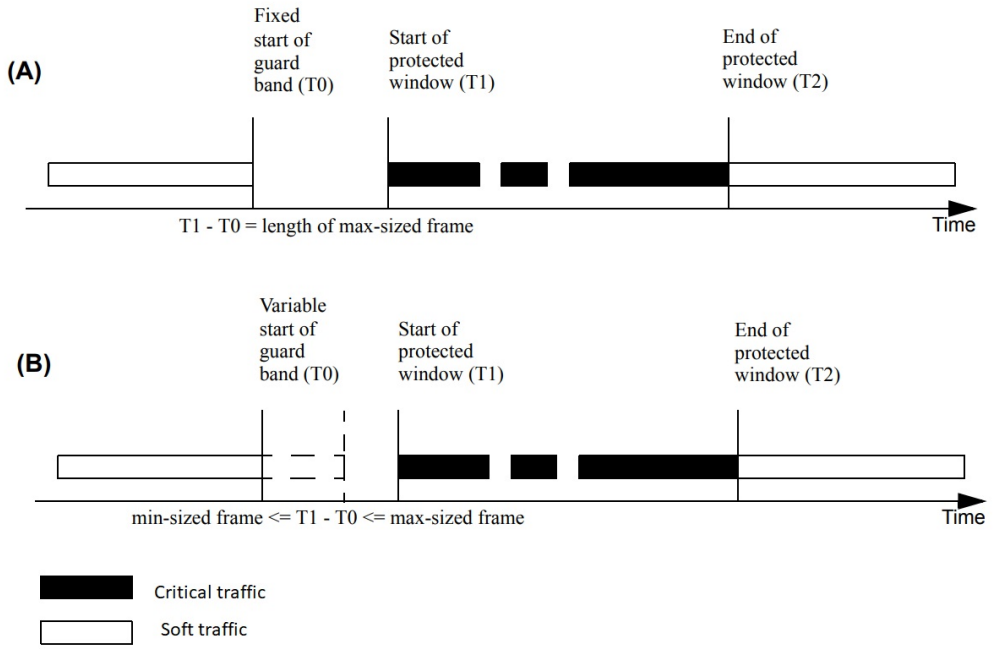


Figure 2.7: Establishing a protected window [5]

and the procedure continues until the last element of the *gate control list* and then it starts again from the first element. In the case that many gates are opened at the same time then the Transmission Selection Algorithm selects the one with the highest priority.

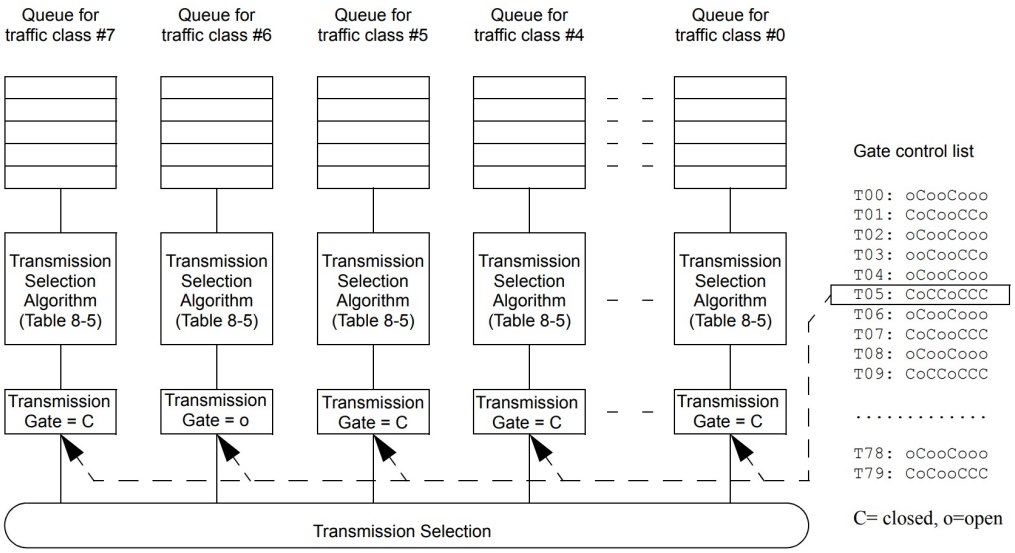


Figure 2.8: Transmission selection with gates [5]

This mechanism implements the protected window and the guard band does not need to be specified since it is implicit in the mechanism. In order to create a protected

window, we must ensure that the gate associated to the desired traffic class is opened during the required time to transmit all the frames. This behaviour is depicted in Figure 2.9. At time T1 the gate associated to the critical traffic (class 3 in this case) is set to open while the others are set to closed, so during the indicated *TimeInterval* only the frames from the queue for traffic class 3 will be transmitted. Then the next parameters of the *gate control list* set the inverse states of the gates so the soft traffic can be transmitted.

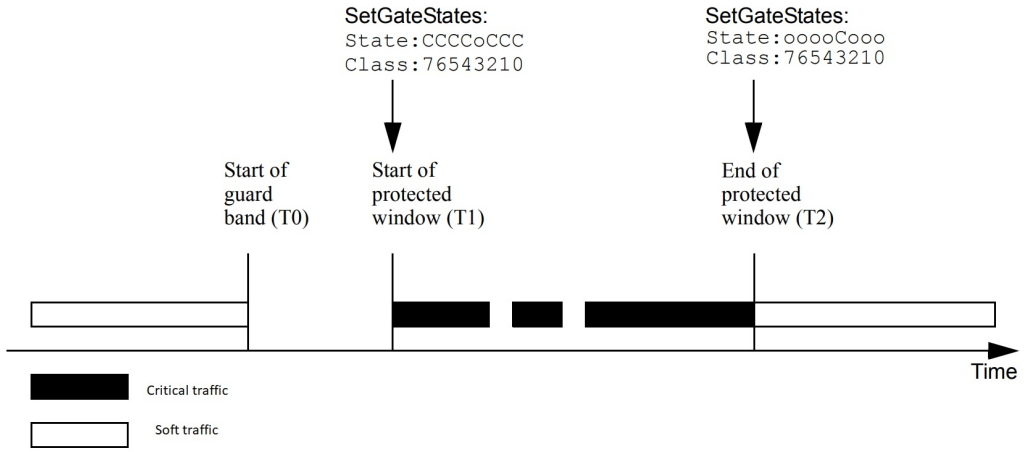


Figure 2.9: Protected window using scheduled traffic [5]

One of the advantages of this mechanism is that it can be implemented on the bridges even though the end stations do not support it.

2.2 NETCONF

To configure the TAS we need to use a network management protocol. The IEEE 802.1Qcc proposes NETCONF, RESTCONF and SNMP from which we have chosen NETCONF.

The Network Configuration Protocol (NETCONF) is a network management protocol from the IETF, first published in 2006 as RFC 4741 and then superseded, in 2011, for the RFC 6241. NETCONF provides mechanisms to install, manipulate and delete the configuration of network devices. NETCONF can be split in the following layers:

1. Secure Trasnport layer establishes the communication between the client and the server. This communication can be achieved using various transport protocols.

2. Messages layer encodes the NETCONF messages.

3. Operations defines the operations that can be used by the client.

4. Content layer is the part that specifies what data is contained in the configuration files and how it is represented. This layer is out of the scope of NETCONF.

Regarding to the Secure Transport layer, NETCONF is not linked to any protocol but the chosen protocol must provide authenticated connections and an appropriate level of security and confidentiality. Two candidates for this purpose are Secure Shell (SSH) and Transport Layer Security (TLS).

NETCONF bases the communication—messages layer— between the client and the server on a Remote Prodecure Call (RPC) model where the network manager is the client and the network device is the server. In this RPC model exists four elements:

- <rpc>: it is the element used to send requests, that correspond to the NETCONF operations, from the client to the server but not from the server to the client; the <rpc> element must contain an attribute called "message-id" that identifies the <rpc>

- <rpc-reply>: it is the element used by the server to respond to a <rpc> from the client; the <rpc-reply> has a mandatory attribute "message-id" that corresponds with the one from the <rpc> to which is responding.

- <rpc-error>: it is the element enclosed in a <rpc-reply> element if there has been some error when processing the <rpc> request.

- <ok>: it is sent in the <rpc-reply> indicating that request has been successfully processed.

More than one <rpc> can be send by the server before receiving the <rpc-reply> but the process must be serial, thus the server will reply the <rpc> in the same order that has been received.

In NETCONF, the configuration of a network device is stored in what is called *configuration datastores*. NETCONF defines the <running> configuration datastore, which contains the current configuration of the device, as mandatory. Additionally, NETCONF defines two optional configuration datastores, the <startup> and the <candidate>.

The <candidate> datastore contains a configuration that can be modified without altering the configuration form the <running> datastore; using the <candidate> datastore the manager of the network can make as many changes as it requires before the configuration is set in the <running> datastore. To stablish the <candidate> configuration as the <running> configuration exists the <commit> operation.

The <startup> configuration datastore has a configuration that will be copied to the <running> datastore at the beginning. To modify the contents of the <running> or <candidate> datastores NETCONF provide the operations <edit-config> and <copy-config> whereas for applying a new configuration in the <startup> datastore it only can be used the <copy-config> operation. Besides, there are operations to retrieve the configuration that is in the datastores.

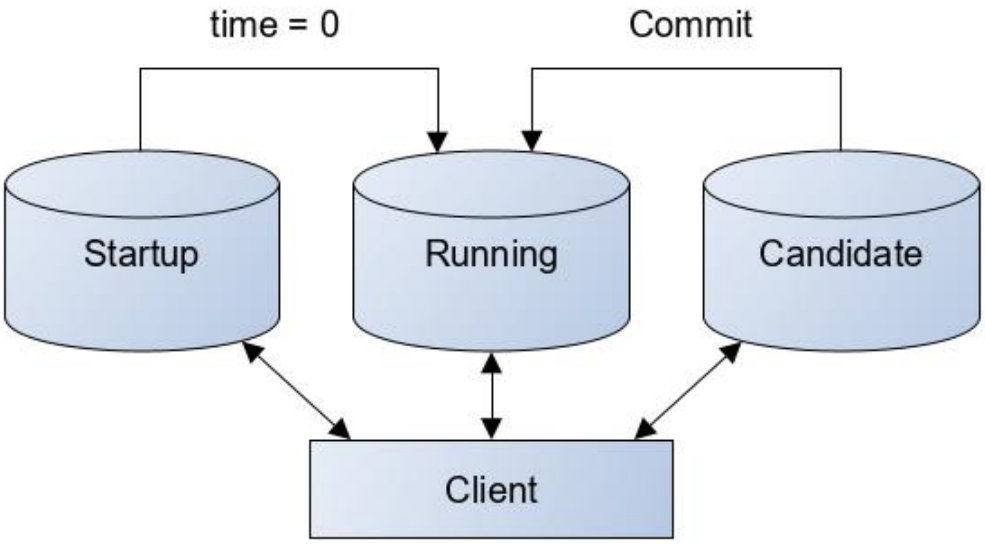


Figure 2.10: NETCONF configurations datastores

As we have already seen, NETCONF provides operations to configure the network devices and retrieve state information. The NETCONF operations are enclosed in the <rpc> requests. The <rpc> indicates to the server which actions it has to perform. NETCONF distinguishes between two types of operations, the *base operations* and *new operations*. Subsequently there is a description of the *base operations* and operation <commit>.

- get-config operation retrieves the specified configuration datastore or all if a filter is not specified.
- edit-config operation loads the specified configuration to the specified target configuration datastore.
- copy-config operation replaces the current configuration datastore of the target with values of the specified configuration. If the target datastore does not exist then it is created.

- delete-config operation deletes the target configuration datastore as long as the target is not the <running> datastore.
- lock operation disallows other NETCONF sessions to make changes in the target datastore. Furthermore, the server must ensure that the locked configuration can not be modified using operations from other management protocols.
- unlock operation releases a configuration that has been previously locked with the <lock> operation.
- get operation retrieves <tunning> configuration and information about the state of the device. The main difference between <get> and <get-config> is that the <get> operation does not allow a target and always retrieve the information from the <running> datastore
- close-session operation terminates the NETCONF session and unlock any locked datastores.
- kill-session operation forces the termination of a NETCONF session. The <kill-session> halt any operation that is being carried out and close all the associated connections. This operation can not terminate its own session.
- commit operation sets the candidate configuration as the running configuration.

Some of the operations described above take one or more parameters. The table 2.2 shows the parameters of every operation where an asterisk is used to indicate the parameters that are optional.

Operations	Parameters
get-config	source, filter*
edit-config	target, default-operation*
copy-config	target, source
delete-config	target
lock	target
unlock	target
get	filter
close-session	-
kill-session	session-id
commit	-

Table 2.2: Parameters of NETCONF operations

The possible values and the meaning of the NETCONF operation parameters,depicted in table 2.2, might be slightly different between different operations but in general terms they are used as follows:

- *Source* parameter indicates from which datastore or file the configuration will be retrieved. The source parameter can take the value startup, running, candidate or the direction of the file containing the data.

- Target is the datastore where the changes will be applied. The target parameter can take the value startup, running or candidate.

- *Filter* parameter indicates which parts of the device configuration will be retrieved and it follows an XML subtree filtering mechanism.

- *Default-operation* parameter selects the operation for the current <edit-config> request. The default-operation parameter can take the values merge, replace, create, delete and remove. If it is no specified the default value is merge. The merge value indicates that the current configuration is modified in order to have the same values as in the configuration from the target parameter but if values from the current configuration are not specified then they will remain as they are.

- *Session-id* parameter indicates the identifier of a NETCONF session that will be terminated. The value can not be the identifier of the current session.

Once explained the basis of NETCONF, we can introduce RESTCONF. RESTCONF is a network management protocol, based on NETCONF, that uses HTTP. The usage of HTTP, makes RESTCONF suitable for Web applications that require access to the configuration data. One important feature is that RESTCONF is compatible with NETCONF so that the datastores from NETCONF can be used in RESTCONF.

2.3 YANG model

NETCONF is a network management protocol, but as we have seen it does not specifies how to model the configuration and the state data. In order to tackle this issue the TSN standards suggest using YANG, a data modeling language to be used by the NETCONF protocol. YANG is described in the RFC 6020.

In YANG, data models are structured into modules and submodules. A YANG module organizes the data as a tree where each node has its own name and a node can have a value or child nodes. A YANG module can expand its hierarchy adding nodes belonging to other modules trough the *import* statement or the *include* statement for the submodules.

The YANG modules have their own equivalence in XML so transforming a YANG into XML makes it suitable to be used by XML applications.

A NETCONF server must implement at least one module but it may implement more. A module is divided in three statements: the module-header, the revision and the definition. The module-header identifies the module and gives information about it. The revision contains the version of the module. The definition statement corresponds with the body of the module where the nodes are defined.

The hierarchy of YANG data is structured in nodes. There are four types of nodes:

- *Leaf nodes* are simple data that only contain one value of a specific type. This type can not have child nodes.

- *Leaf-list nodes* are a sequence of leaf nodes and therefore, each leaf only can contains one value of a particular type.

- *Container nodes* are used to group related nodes. It only contains child nodes and it is mainly used for organising the hierarchy of the data.
- *List nodes* are a sequence of list entries that can be any of the previous nodes mentioned. A list node is identified by its *key*.

A *leaf node* can take a value of a certain type. YANG defines a set of types[2.3] for the data that is indicated using the statement *type*.

Name	Description
binary	Any binary data
bits	A set of bits or flags
boolean	"true" or "false"
decimal64	64-bit signed decimal number
empty	A leaf that does not have any value
enumeration	Enumerated strings
identityref	A reference to an abstract identity
instance-identifier	References a data tree node
int8	8-bit signed integer
int16	16-bit signed integer
int32	32-bit signed integer
int64	64-bit signed integer
leafref	A reference to a leaf instance
string	Human-readable string
uint8	8-bit unsigned integer
uint16	16-bit unsigned integer
uint32	32-bit unsigned integer
uint64	64-bit unsigned integer
union	Choice of member types

Table 2.3: YANG types

YANG defines several statements, which can be found in the RFC 6020, and the most relevant for this Bachelor Thesis are described below:

- Namespace statement defines the XML namespace.
- Prefix statement defines a prefix that is associated with the module. The prefix can be used to make reference to that module.
- Revision statement indicates the version of the module.
- Default statement indicates the value that will be used if the value is not defined.

As we mentioned before, YANG has its equivalent in XML. To obtain the XML from the YANG data model, YANG describes the mapping rules that must be followed. These rules are detailed in the YANG specification [11].

CHAPTER 3

PHASES AND TIMELINE OF THE PROJECT

The execution of the project can be split in different phases. These phases represent the different parts of the Waterfall model. The Waterfall methodology is a type of software development which consists in a sequential development, it is divided in phases that do not overlap so one phase must be completed before executing the next one. Below, there is a diagram depicting the phases of the project and the corresponded phases of the Waterfall methodology:

The phases match a Waterfall model with the following equivalences:

- The *requirements* are represented as the *Study of the relevant TSN specifications*, due to it is where the specifications are examined in detail.
- The *system design* corresponds to the *design* whose objective is specify the overall system architecture.
- The *implementation* is covered by two phases, *Search and selection of designing tools* and *Implementation and testing of the mechanisms*.
- The *integration and testing* corresponds to the *Integration and validation*.
- Regarding to the *deployment of the system* and the *maintenance*, they have not been achieved because of the restrictions exposed in section 4.1. Therefore, these parts should be included in a future work

The *study of the technology* consists in the investigation of the technologies, standards and protocols that were needed to realize the project. The that has been done in this phase is represented in the chapter *Study of the relevant TSN specifications*.

In what refers to the *Design*, it is the part where the decisions involving to the hardware and software have been taken. This phase is remarkably important, seeing that several subsequent phases are based on it.

The *Search and selection of designing tools* phase explains the tools that have been chosen to execute the implementation and the *Implementation and testing of the mechanisms* describes how the output of the previous phase is used in order to execute

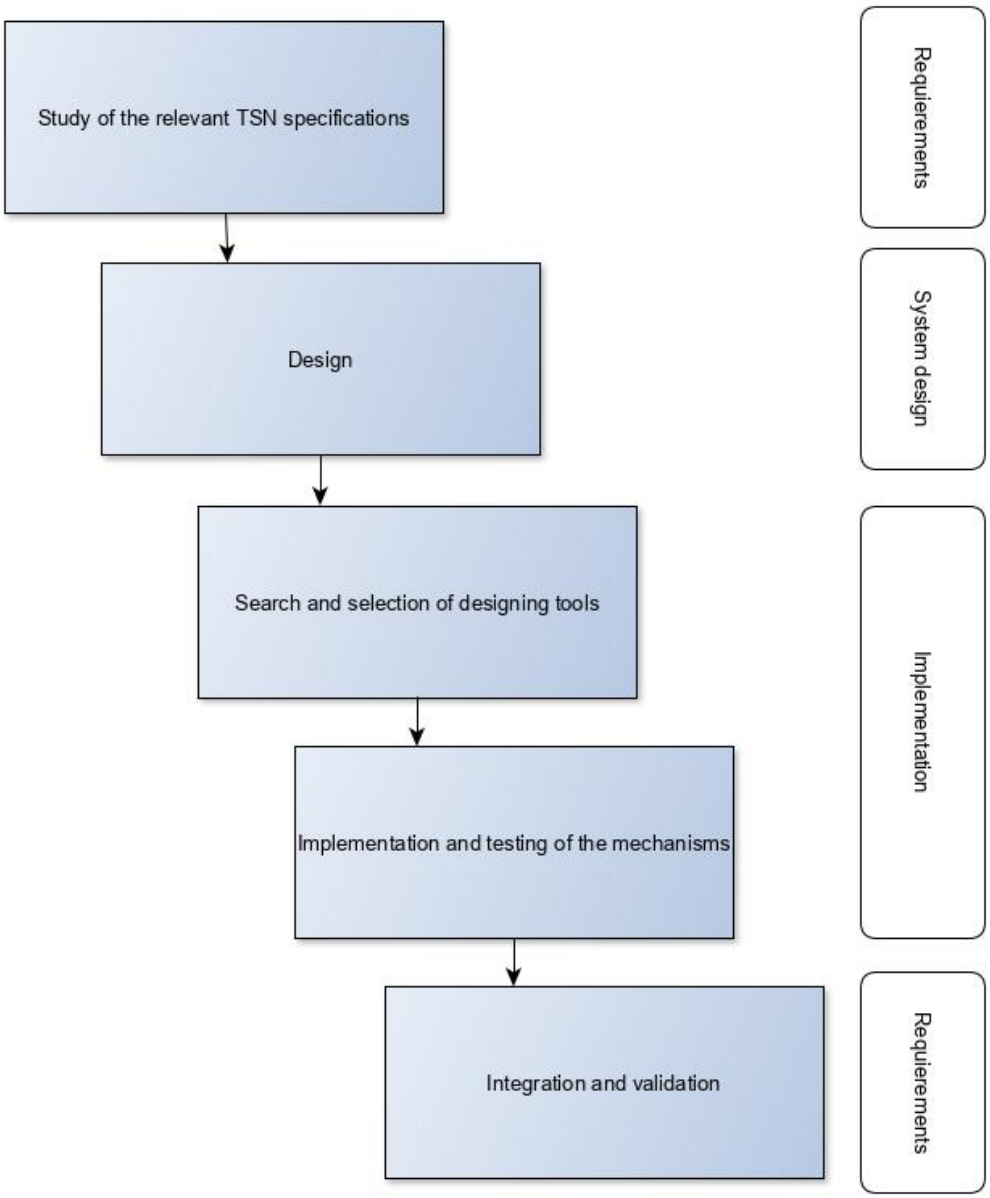


Figure 3.1: Correspondence between project phases and waterfall phases

the implementation. Even though in the figure 3.1 may looks like one phase is subsequent to the other, we must recall that both are inside the implementation so iterations between these phases can exist and will exist for the reasons explained hereunder.

Finally, the Integration and validation is intended to incorporate the implemented mechanisms, in the phase *Implementation and testing of the mechanisms*, into the system and ensure their correctness.

Concerning to the duration of the Bachelor thesis duration, it was started at the beginning of the first semester and has lasted until the end of the academic year. The figure 3.2 shows the invested time in each phase and the writing of the report. During the phases *Search and selection of tools* and the *Implementation* we found several problems due to the libraries and the connection with the switches and also with the switches, consequently these have been the phases that have taken more time during the execution of the project.

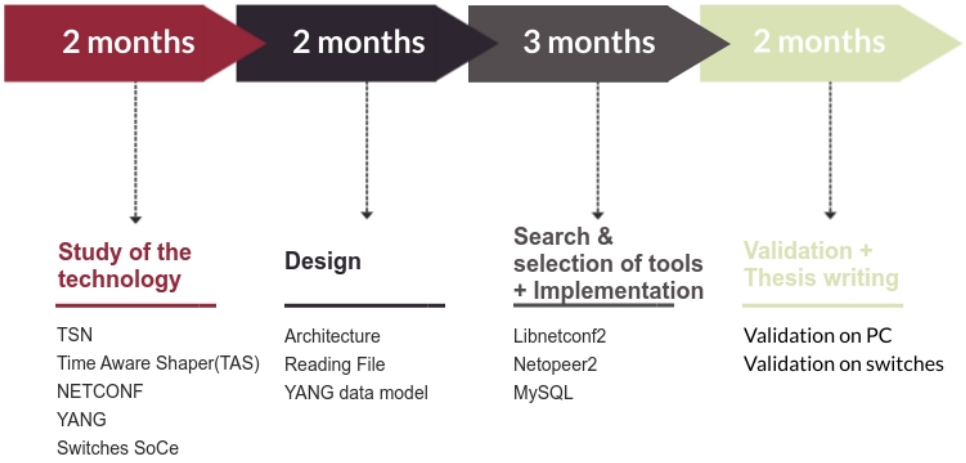


Figure 3.2: Timeline of the project

CHAPTER 4

DESIGN

The current chapter is devoted to give an overall view of the initial conditions and justify the decisions we made concerning the design of the hardware and software architectures.

4.1 Starting point

We should note that at the beginning of this work we counted with some already existing hardware and software platforms. We now describe these platforms, to better understand the decisions we made in the design of our architecture.

4.1.1 Hardware

In this project we count with two SoC-e SMART MPSoC switches (Figure 4.1). The SOCe SMART MPSoC is a switch that supports the main TSN standards. Specifically, the TSN standards incorporated in the board are:

- IEEE 802.1AS-2011: This standard basically provides the synchronization mechanisms.
- IEEE 802.1Qbv: It describes the TAS which enables scheduled (time-triggered) traffic that allows a deterministic behaviour.
- IEEE 802.1Qav: It defines the Credit Based Shaper that allows the definition of the maximum fraction of the bandwidth that is available to a determined queue.
- IEEE Std 802.1Qcc: It provides mechanisms to configure and manage the network resources in a way that scheduled traffic is supported.

A more detailed description of these standards can be found in the chapter Technical Background(2).

The main elements of the board, depicted in Figure 4.2, are:

- TSN Traffic Generator: it makes possible the generation of TSN traffic streams. The TSN Traffic Generator is composed by two parts. The first defines the parameters of the standard Ethernet frames whereas the second is in charge of the TSN configuration parameters.
- TSN Adapter: It allows a standard non-TSN device to transmit TSN traffic.
- PORT-Z: This port is used to configure the board.
- MTSN Switch IP: The IP implements the actual TSN switch. It is composed by six ports, four external (Port-0 to Port-3) and two internal ports,one connected to the TSN adapter and the other to the TSN Traffic Generator.

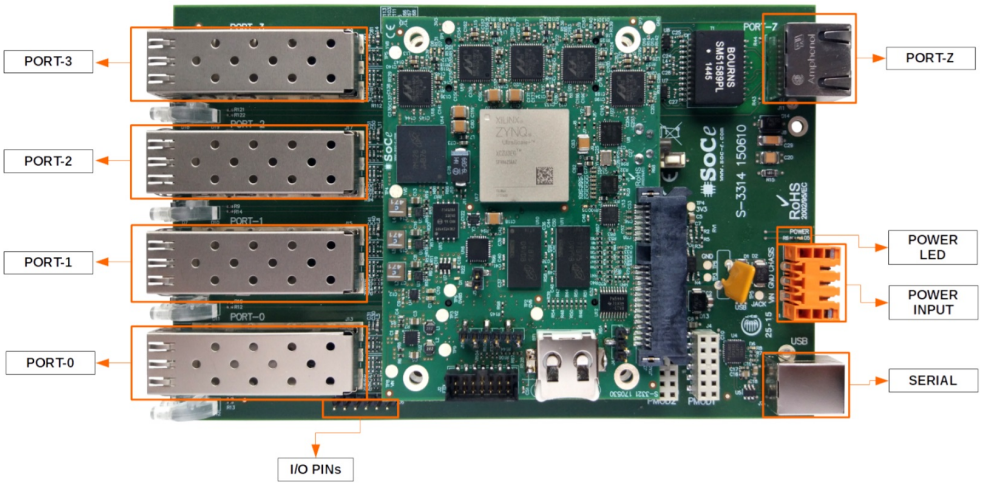


Figure 4.1: Switch SOCe SMARTMPSoC [6]

The switches have a default IP configuration that has not been modified in this project. Table 4.1 shows the IPs of the switches.

switches IP (depending on the board)
192.168.4.64
192.168.4.65

Table 4.1: Switches IP

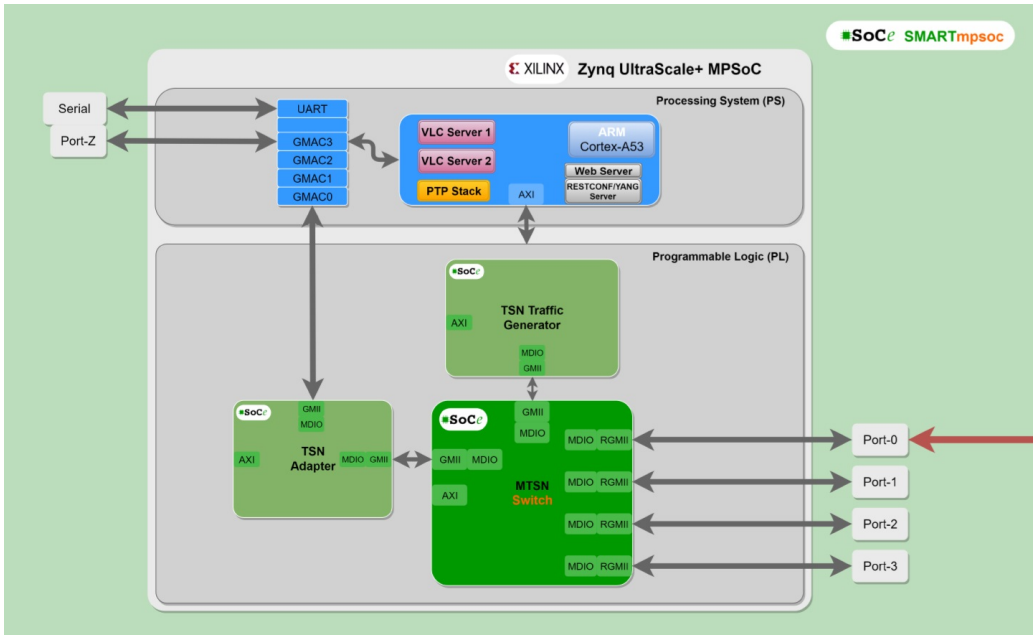


Figure 4.2: SOCe SMARTMPSoC diagram[6]

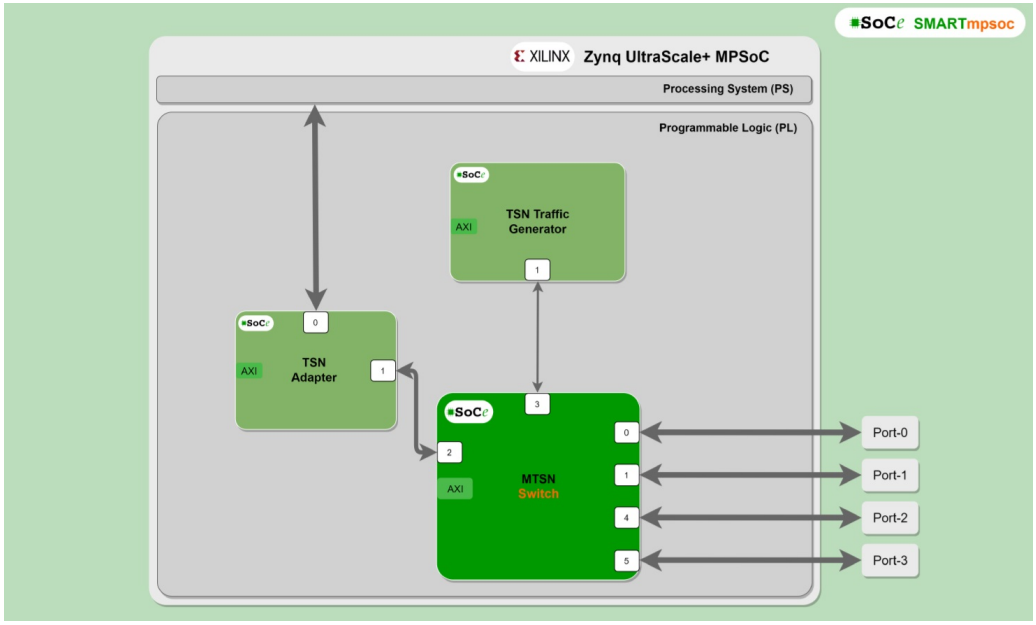


Figure 4.3: SMART MPSoC ports[6].

4.1.2 Software

During this project we counted with the software provided by the vendor (SoCe), which consists in a graphical interface that allows modifications on the different parameters of the switch. The network management protocol used by this software is RESTCONF and thus, the communication in the network is done using HTTP.

One of the functionalities provided by this software is the configuration of the TAS.

Specifically, this configuration is done manually using a graphical interface provided by the manufacturer. Thus, this configuration can not be automitezed, but must be done by the user. We next show the steps that must be followed to configure the TAS through the graphical interface.

1. As we mentioned, the protocol used to transmit the configuration to the switches is HTTP. Thus, we can access the user interface using any web browser. Specifically, in the web browser we must type the IP addess of the device we want to configure, followed by the configuration port, which is 3333. Once we access the board, we see the configuration menu shown in Figure 4.4.

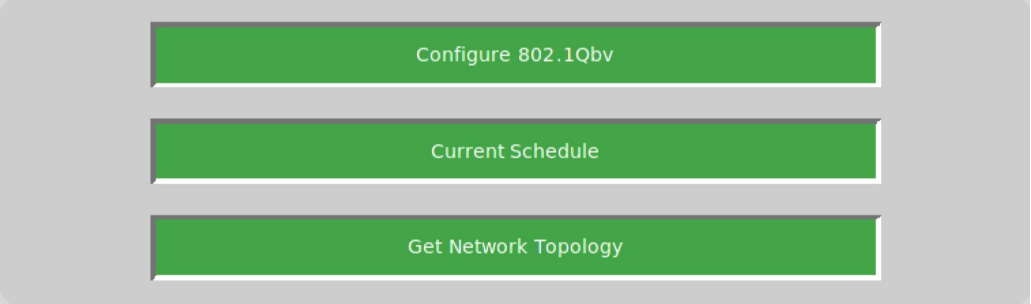


Figure 4.4: Menu of graphical interface to configure TAS.

2. Once in the menu, we select the option to configure the TAS (Configure 802.1Qbv); after that different menus appear one after the other to introduce the different parameters of the TAS.

3. The first menu—Generic parameters—configures two parameters: the "Window Length" that corresponds with the hypercycle and the "Number of Time Slots" that refers to the quantity of gates-state. In Figure 4.5 the values 90us and 8 have been selected for the "Window length" and "Number of Time Slots" respectively. Once we have chosen the configuration we select continue.

4. The next menu, Time Slot Parameters(Figure 4.6), allows us to define the duration and the the gates-state of each slot. When the parameters have been introduced we can press continue.

5. The last menu, Device Selection, allows to selects in which ports from which switches the configuration will be uploaded. In Figure 4.7 there is an example of how to send the configuration to several ports of different switches. Basically the steps followed are select the desired device(i) and then the ports(ii); repeat the previous step as many times as necessary(iii) until all the desired ports have been selected and finally select the option "send configuration"(iv).

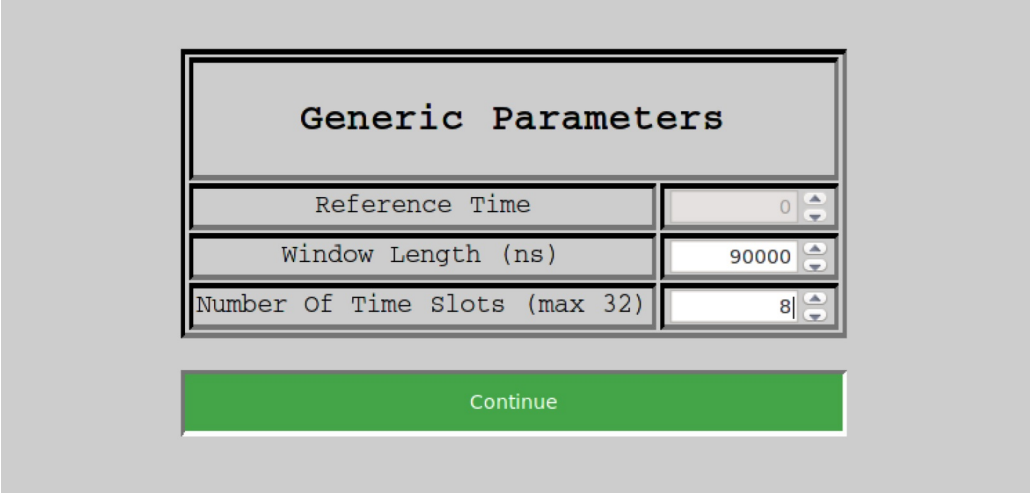


Figure 4.5: Generic Parameters menu, configures the hypercycle and defines the number of gates-state.

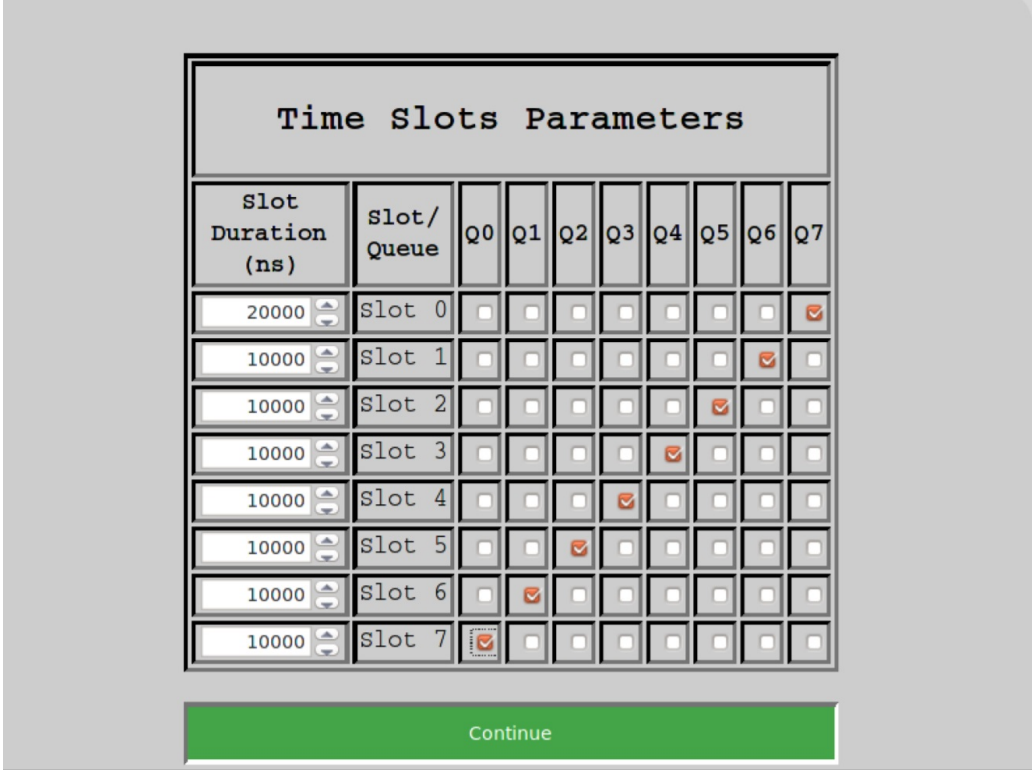


Figure 4.6: Time Slots Parameters menu, which allows to indicate the period and gates-state of each slot.

As we can see, configuring the TAS using the graphical interface is a long and tedious work. This is specially true when we have different configurations for each port and device, as the whole process (steps 1 to 5) must be done for each configuration. Moreover, in complex networks with complex schedules the number of slots will be

(a) Configure ports 0 and 1 of the device with IP 192.168.4.66.

(b) Select a new device to apply the same configuration.

(c) Configure ports 0 and 4 of the device with IP 192.168.4.67.

(d) Transmit the configuration to all the selected devices.

Figure 4.7: Steps in Device Selection Menu to set the configuration in the desired ports.

significantly higher. For these reasons, in this work we provide a mechanism that allows to automatize the configuration of the TAS.

4.2 Design of the architecture

The architecture of the automatic configuration system specifies the different parts of it, the functions of each part and the relation between them; therefore it describes the interrelation of the hardware with the software. We must note that TSN already defines a series of network architecture models. Thus, we selected the one that better adjusts to our goals.

The amendment IEEE Std 802.1Qcc 2.1.4 specifies different architecture models to organise the different devices of the network and the role that they play in the network communication. From these proposed architectures, we have chosen the *Fully centralised model* and discarded the others for the following reasons:

- The *Fully distributed model*, defined in the IEEE 802.1Qcc, is based on that that the network configuration is done in a distributed manner in all devices. However, this behaviour does not allows the implementation of the TAS and it is only suitable to configure the Credit-based shaper so we have discarded this option.

- The *Fully centralised model* incorporates CUC over the CNC in order to execute computational complex operations needed to generate an appropriate configuration for the end stations. This model is suitable for implementing the TAS but since we do not have the objective of configuring the end stations we have decided to not select the *Fully centralised model*.

- Finally the *Centralised network/distributed user model* is devised to use a CNC, which has a complete view of the physical network topology, for configuring the network using a network management protocol. As the *Fully centralised model*, it allows to configure the TAS. We want to implement the configuration of the TAS without computing its parameters. Since we do not need the requirements of the network the CUC over the CNC it is not useful. For this reason, we have decided that the *Centralised network/distributed used model* is the best approach to achieve our goal.

The CNC is responsible for computing the gate control list given the necessities of the network. However, the generation of a gate control list is out of the scope of the project; for this reason, our design is based on the premise that a user or a scheduler[1] will provide us a gate control list that has already been generated to satisfy the network requirements. In the figure 4.8 it is represented the relation between the user/scheduler, the CNC and the switches.

The standard suggests to use as a network management protocol to configure the network, such as SNMP, RESTCONF or NETCONF. SoCe uses RESTCONF for configuring the switches and the data is structured with a compatible YANG model. Since RESTCONF is oriented to Web applications and is not as secure as NETCONF that uses SSH we have decided to use NETCONF for our mechanism.

As a result of selecting NETCONF, our architecture to manage the network is based on a client/server architecture where the CNC and the switches take the role of client and servers respectively. To configure the network, the CNC (client) will send a request

[1] The scheduler would represent the part of the CNC in charge of computing the gate control list that fulfils the requirements of the network

specifying the configuration to be set and the switch (server) will respond informing whether the configuration has been set or not.

The IEEE Std 802.1 Qcc defines that the CNC can exist in an end station or in a bridge but the problem is that such implementation would require specific hardware and it is not economically available for a vendor because the standards are continuously being updated. Thus the implementation may not be compatible with the new version of the standard. Consequently we have decided that the CNC will be implemented on a computer.

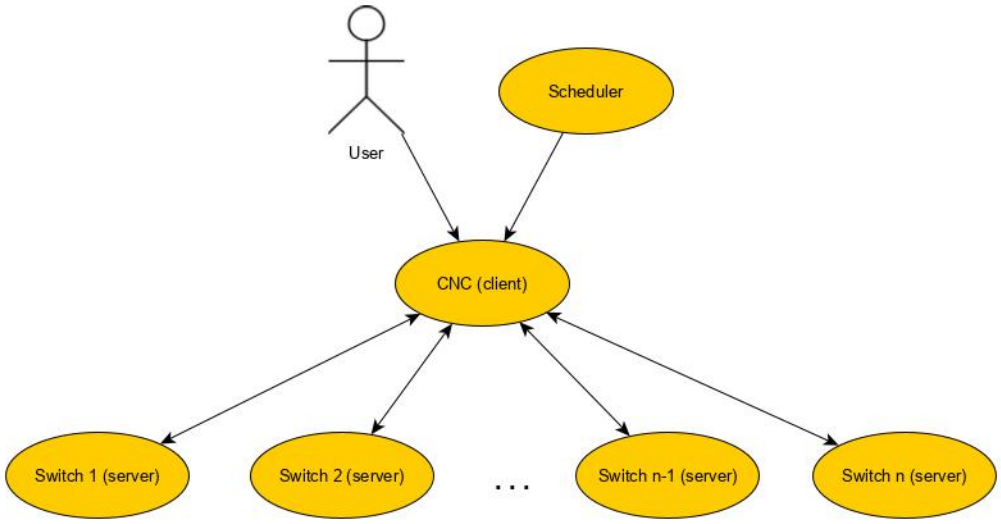


Figure 4.8: *Centralised network/distributed used model* depicted in our system

In order to set the TAS configuration, the user has to send the values of the parameters to the CNC(client) and indicate in which ports of which switches he wants to deploy that configuration. To receive the TAS configuration from the user, we propose a reading file where all the needed information will be specified.

To model the configuration to be set using NETCONF, we have decided to use YANG because it has been created for this purpose. The P802.1Qcw standard that will define the YANG model for *Time Aware traffic Class* but it is still under development. This means that a YANG model for *Time Aware traffic class* has not been published yet, hence we have to design our own YANG model. On account of the fact that we are using YANG, we need to generate its XML representation, called *configuration file*, that will be used to modify the values of the YANG data model parameters in the *TAS database*.

Additionally, we want the client to store configurations if it is necessary. To add this capability to the system we have introduced the *user database* in the client-side that will be implemented to be used by a user. Figure 4.9 shows the elements of both server and client.

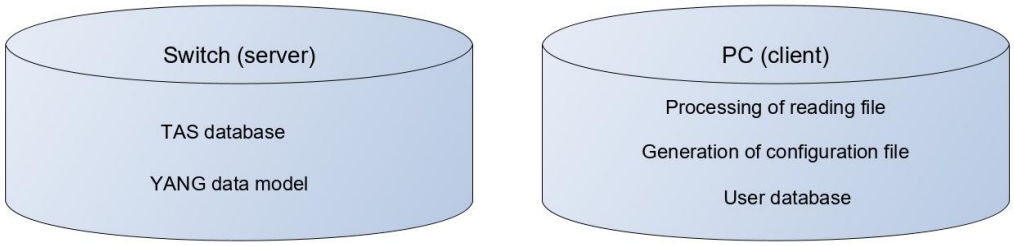


Figure 4.9: Elements of the client and the server

4.3 Design of the network configuration procedure

In the previous section we have described the architecture model we use and the elements that make it up; taking this into account now we proceed to specify the steps needed to configure the network.

Figure 4.10 depicts the procedure to configure the network. The initialisation of the system consists in running the servers, with a data model of the TAS, in the switches. The data model of the configuration is a YANG model. As long as the initialisation in the server-side has concluded, the client-side can start running. Then the server can receive requests from the client and perform actions in consequence.

In the client-side the procedure is as follows:

1. The user uses the console to indicate to the client which actions it has to carry out. In this part the user has several options: (1)configure the ports with configurations provided by the user,(2)configure the ports using a configuration from the *user database*, (3) store a new configuration in the *user database* or (4)delete configurations from the *user database*.

2. If in the previous step the user has selected the options 3 or 4 then the corresponding operation will be carried out and we will go back to the first step. For the option 3, the client need to get the configuration parameters. Towards this end we need to establish how these parameters will be transferred from the user to the client, this is achieved through a reading file (see section 4.4). If the options 1 or 2 are selected then it will proceed with the next step.

3. For the options 1 and 2 we need to obtain the configuration parameters from the user or the *user database* respectively. If the chosen configuration belongs to the *user database* then the information will be retrieved from it; for the case that the configuration comes directly from the user, it will have to provide a reading file as the one mentioned for storing new configurations in the *user database*.

4. Once the configuration parameters have been obtained, the client has to generate the XML file corresponding to the YANG model in the server.

5. The XML file generated is used to create an RPC that contains the operation edit-config. After creating the RPC, it is sent to the server(switch).

6. Finally the server receive the request from the server and set the configuration in the *TAS database*.

This procedure will be repeated every time the user wants to configure the TAS.

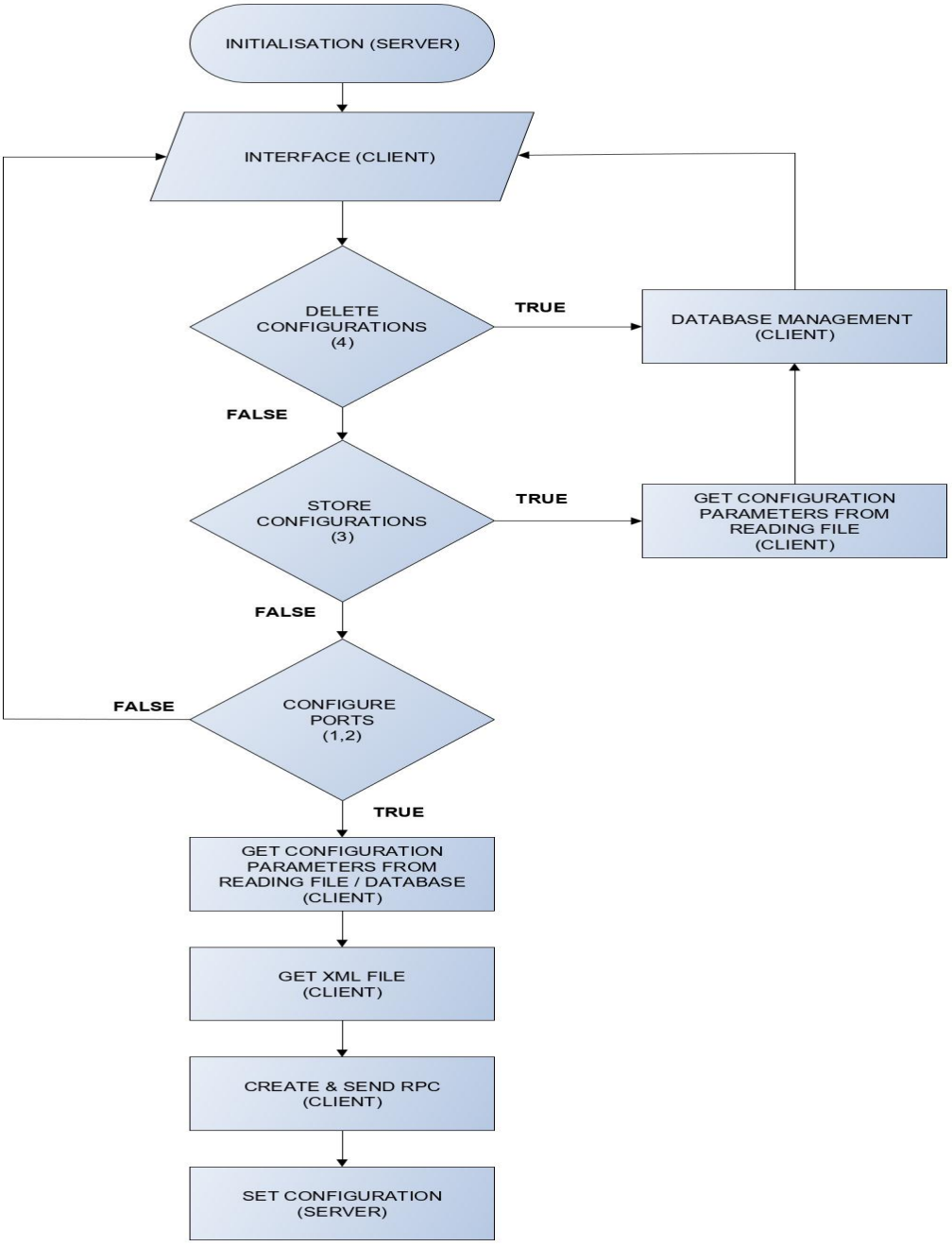


Figure 4.10: Flux diagram of the configuration procedure.

4.4 Design of the reading file

The reading file has the purpose of allowing the user to send one or more gates control lists to the CNC (client) for further processing.

The reading file must contain the gates-state and the period for each gates-state. In addition it must appear to which port of which switch every gate control list corresponds. Given this conditions we have designed the reading file as is shown in Listing 4.1.

```
switch_number: switch-number(0)
  port_number: port-number(0)
    period(0),gates-state(0)
    period(1),gates-state(1)
        ...
    period(m-1),gates-state(m-1)
    period(m),gates-state(m)
  port_number: port-number(1)
    period(0),gates-state(0)
    period(1),gates-state(1)
        ...
    period(m-1),gates-state(m-1)
    period(m),gates-state(m)
  ...
  port_number: port-number(i)
    period(0),gates-state(0)
    period(1),gates-state(1)
        ...
    period(m-1),gates-state(m-1)
    period(m),gates-state(m)
switch_number: switch-number 1
    ...
switch_number: switch-number n
```

Listing 4.1: Format of the reading file

The format follows a tree structure and it is composed by the following elements:

- switch number: it is the address of the switch that has to be configured.
- port number: it is the port that has to be configured.
- period: is the time, in nanoseconds, that the gates-state will be used until the next gates-state is set.
- gates-state: indicates the state of each gate of the port. The port has 8 different traffic classes, and therefore 8 gates that are represented using with a binary number of 8 digits. Each gate corresponds with one traffic class. Following the nomenclature of the Section 2.1.5,the most significant bit represents the traffic class 7 and the less significant bit the traffic class 0. As explained in the Section

2.1.5 a gate can be in one of two possible states, open or closed. The open state is represented with a '1' and the closed state with a '0'.

The main criterion considered when designing the format of the reading file, has been the clearness of it so a human could read and modify the file easily.

The reading file must contain the gate control list of every port that is configured. If there is a port that is not affected by the new configuration, then it does not have to be included in the list. Regarding to the switches, a switch can appear several times in the same reading file with different ports and gate control lists. The description of the process that is done with the information of the reading file is in Section 6.1.

Listing 4.2 shows an example of a reading file. In the example below, there are two switches being configured. One of them appears twice in the file with different ports and gate control lists. This is used to store more than one configuration of the switch in the *user database*.

```
switch_number: 64
  port_number: 0
    20000,10000001
    10000,00000010
    30000,10000010
  port_number: 1
    40000,00010000
    10000,10000000
  port_number: 3
    40000,10010000
    20000,01000001
    10000,10000001
switch_number: 65
  port_number: 0
    10000,00010000
    10000,00100000
    10000,10000000
  port_number: 4
    10000,00000111
    20000,10001000
    10000,10001111
switch_number: 64
  port_number: 1
    10000,00000001
    20000,00000010
  port_number: 3
    40000,00010000

    30000,11000000
    10000,10000000
```

Listing 4.2: Example of reading file

4.5 Design of the YANG model

The software architecture is formed by a client (CNC) and a server (switch) that use NETCONF for the communication. To model the configuration of the TAS we have decided to use a YANG model, for the reasons explained in the previous section 4.2.

To create the correct YANG data model we had needed to get parameters needed to configure the network. The switches already implement a server, different from the one we use, from which it has been obtained the parameters that are used for the configuration; with the graphical interface a configuration has been set and then looking into a log file—created by the server and containing the configuration parameters— we obtained the information that the YANG data model must contain.

Listing 4.3 represents the information— using the JSON format — that the YANG data model must contain. The parameters highlighted in blue can be configured while the other information must remain as it is depicted.

Listing 4.4 shows the YANG data model to be used. We define a module named NewSwitch which contains: a namespace that defines the XML namespace that must be used when modifying the data; a prefix ("bld") that is associated with the module and its "namespace". The data model is defined inside the body of the module,which is the part subsequent to the module statement. The module statement is represented with the information inside the brackets of module NewSwitch.

To organise the data in the YANG model we have used the nodes specified in Section 2.3. As we can see, the hierarchy of the information represented in Listing 4.3 is organised using lists. In those cases where a list only contains other lists we represent them in the YANG as containers since they do not contain any value. P.e the list "ietf-interfaces:interfaces", only contains the list "interface" thus it must be a container. For this reason, the following nodes have been defined as "containers" too: "ieee802-dot1q-bridge-bridge-port", "ieee802-dot1q-sched-gate-parameters", "container admin-cycle-time", "sgs-params" and "admin-base-time". For those lists that have a value we have used the statement "list" that allows multiple instances in the data tree; to identify the list a key is defined. The end nodes allocated inside the lists are defined as leafs of the corresponding type; for those end nodes whose value can not be modified we have used the statement default to assign the value that will have if it is not specified. In addition, some leafs contain the statement "units" that contains a textual definition of the units associated with the end node.

```
"ietf-interfaces:interfaces": {
  "interface": [
    {
      "type": "ethernetCsmacd",
      "enabled": true,
      "name": "port-number",
      "ieee802-dot1q-bridge:bridge-port": {
        "ieee802-dot1q-sched:gate-parameters": {
          "admin-cycle-time": {
            "denominator": 500/hypercycle,
            "numerator": 500
          }
          "config-change": true,
          "admin-control-list": [
            {
              "sgs-params": {
                "time-interval-value": period,
                "gate-states-value": gates-state
              },
              "index": 0
            },
            {
              "sgs-params": {
                "time-interval-value": period,
                "gate-states-value": gates-state
              },
              "index": 1
            },
            ...
            {
              "sgs-params": {
                "time-interval-value": period,
                "gate-states-value": gates-state
              },
              "index": n-1
            }
          ],
          "admin-control-list-length": n,
          "admin-base-time": {
            "seconds": "0",
            "fractional-seconds": "0"
          }
        }
      }
    },
    {
```

```
      "type": "ethernetCsmacd",
      "enabled": true,
      "name": "port-number",
      "ieee802-dot1q-bridge:bridge-port": {
        ...
      }
    },
    ...
  ]
}
```

Listing 4.3: YANG data model of the TAS parameters using JSON encoding. The parameters that can be configured through NETCONF are highlited in blue.

```
module NewSwitch {
  yang-version 1;
  namespace "urn:NewSwitch:test";
  prefix bld;
  organization "UIB"
  contact "Toni";
  description "yang␣model␣for␣TAS"
  revision "2019-04-10"{
    description "final␣version";
  }

  container ietf-interfaces-interfaces{
    list interface{
    key name;
    leaf type{
      type string;
    default ethernetCsmacd;
    }
    leaf enabled{
      type boolean;
    default true;
    }
    leaf name{
    type uint32;
    }

    container ieee802-dot1q-bridge-bridge-port{
    container ieee802-dot1q-sched-gate-parameters{
      container admin-cycle-time{
        leaf denominator{
        type decimal 64{ fraction-digits 6;}
      }
      leaf numerator{
```

```
        type uint 32;
        default 500;
      }
      }
      leaf config-change{
        type boolean;
      default true;
      }
      list admin-control-list{
        key index;
      container sgs-params{
        leaf time-interval-value{
          type uint32;
        units nanoseconds;
        description "time␣that␣the␣gate␣configuration
␣␣␣␣␣␣␣␣must␣be␣held";
        }
        leaf gate-states-value{
        type uint32;
        units binaryRepresentation;
        description "configuration␣of␣the␣gates";
        }
      }
      leaf index{
        type uint32;
        description "position␣of␣the␣state␣in␣the
␣␣␣␣␣␣␣␣gates␣control␣list";
      }
      }
      leaf admin-control-list-length{
      type uint32;
      }
      container admin-base-time{
      leaf seconds{
        type uint32;
        default 0;
      }
      leaf fractional-seconds{
        type uint32;
        default 0;
      }
      }
      }
    }
  }
  }
}
```

Listing 4.4: YANG data model for the TAS in switches SMARTMPSoC

4.6 Design of the user database

The *user database* removes the necessity of having numerous reading files when the user wants to use one configuration more than once. Moreover, a database might be useful in systems that must provide a different service in different moments of its operation, i.e., systems that have different phases during their mission. In these cases, there must be a set of configurations ready to be deployed, each configuration for a different phase of the mission.

Regarding to the *user database* contents, we have decided to store all the configuration parameters except the admin-control-list-length that is easily obtained by adding 1 to the last index.

Table 4.2 shows all the parameters from the data model configuration (except from the admin-control-list-length), the switch-number and a new element that is the list-number. The list-number is used to identify the different gate control lists available for a port. In the same table there is a brief example where for the port 0 of the switch 64 there are two available configurations to be set. This way the user will be able to choose the desired configuration indicating the switch-number, the port-number and the list-number.

Database					
switch-number	port-number	list-number	index	period	gates-state
64	1	0	0	10000	128
64	1	0	1	20000	64
64	1	0	2	10000	36
64	1	0	3	30000	8
64	1	1	0	20000	255
64	1	1	1	10000	4
64	1	1	2	10000	18

Table 4.2: Database structure

CHAPTER 5

SEARCH AND SELECTION OF DEVELOPMENT TOOLS

In the previous chapter we have described the design. However, we did not have any tool where to proceed with the implementation. Therefore, we had to search a tool that satisfies our requirements. Fundamentally, we needed to tools. The first tool had to allow us to use NETCONF operations. The second one had to provide us the mechanisms to manage a MySQL. Besides, both tools had to be compatible with c language.

5.1 Libnetconf2

After a deep research we found libnetconf2, a library in C that implements NETCONF. The main features of libnetconf2 are:

- As it has been explained in 2.2, NETCONF uses a transport protocol that is not fixed but must satisfy a set of requirements p.e it must provide authentication; Libnetconf2 allows the user to choose between two different transport protocols, Secure Shell (SSH) and Transport Layer Security (TLS) by using the libraries libssh or OpenSSl respectively.

- Libnetconf implements the RPC communication including the creation, sending, receiving and replying; all the client base operations described in RFC 6242 are implemented; the table 5.1 illustrates the correspondence between the NETCONF operations and the libnetconf2 functions.

- In addition libnetconf2 implements NETCONF Event Notifications defined in RFC 5277. The NETCONF Event Notifications is an optional capability of NETCONF that allows the sending of asynchronous messages.

– To work with YAND modeled data and schemas libnetconf2 uses the library libyang.

NETCONF operations	libnetconf2 functions
get-config	nc_rpc_get_config()
edit-config	nc_rpc_edit()
copy-config	nc_rpc_copy()
delete-config	nc_rpc_delete()
lock	nc_rpc_lock()
unlock	nc_rpc_unlock()
get	nc_rpc_get
close-session	this operation is not explicitly defined because it is implicitly implemented in nc_session_free
kill-session	nc_rpc_kill

Table 5.1: NETCONF base operations implemented in libnetconf2

The main libnetconf2 lack is that it does not implement a data-store and therefore it must be implemented.

In general terms libnetconf2 allows us to manage a network using NETCONF. The main steps to achieve the RPC communication between the server are depicted in the figure ;as it is seen, the first step in open a channel, using SSH or TLS, where the communication will be carried out; then the client can create a RPC element—using the above functions— and send it with the function nc_send_rpc(). Once a RPC has been sent, the server has to handle it and reply; the reception of the server reply is done by the server through the function nc_recv_reply().

Libnetconf2 seemed to be a good platform where to implement our design so we tried the implementation on it. As it has been explained using libnetconf2 the user is the one in charge of doing the open a channel for the communication between the server and the client; for this purpose we chose SSH and we tried to achieve the connection without succeeding; since the SSH messages are encrypted it is no possible to see the messages that are being exchanged and as a result we could not find the mistake in our communication. As we could not directly use libnetconf2 for our implementation, we decided to seek another library that have ended being Netopeer2.

5.2 Netopeer2

Netopeer2 has been the platform over we have executed the implementation of the project. Netopeer2 is a set tools to configure a network based on the NETCONF protocol. The main aspects of Netopeer2 are:

– Based on the libraries libnetconf2 and libyang. Therefore it has all the features from libnetconf2 (see section 5.1)

- Incorporates a datastore using Sysrepo. In our system, the Sysrepo datastore constitutes the *TAS database*. It stores the configuration of YANG model that has been previously provided to sysrepo.
- Includes a client and a server. The client receives instructions from the command line and then send the RPCs to the server to manipulate the configuration data. The server receives the petitions from the client and executes them.

5.3 MySQL

The candidate decided to include a database in the client-side to store different configurations. The selection of the database have been based in the previous knowledge of the candidate about the subject. Since in the degree there is the subject Informatica industrial where some aspects of relational databases are learned the candidate decided to use the relational database MySQL.

CHAPTER 6

IMPLEMENTATION AND TESTING OF THE MECHANISMS

During the *Design* chapter we have defined the architecture of our automatic configuration system, which incorporates several elements, and then in the *Search and selection of development tools* we have described the chosen platform we have used to implement the design. In the current chapter we describe how we have accomplished the implementation of the mechanisms that have been described in those previous chapters and their correctness.

6.1 Processing of configuration parameters

The reading file (see section 4.4)is where all the configuration parameters are introduced by the user and therefore it is from where we have to take the information to configure the TAS in the switches. The required parameters to configure the TAS correspond to the ones highlighted in blue in Listing 4.3. In the reading file only appears the port-number, period, gates-state but not the hypercycle, index nor admin-control-list-length that is required for each gate control list. We decided not to explicitly include this information since it can be computed from the other parameters. Following there is how we have obtained each parameter:

- hypercyle: it is the total amount of time needed to execute the entire gate control list without taking into account the overheads of the system. As we know the gate control list consist in a list of gate operations that have associated one period. Therefore to calculate the hypercycle we have computed the sum of all the periods of a gate control list.

- index: it is the position,starting from 0, of each element of the gate control list. To determine the index we have stored the position of the gate control list elements while obtaining the parameters from the reading file.

– admin-control-list-length: it corresponds to the number of elements in a gate control list and therefore it can be computed as the higher index plus one, so once we obtain the last index we calculate and store the gate control list.

In order to check the correctness of this mechanism we have tested numerous times, for different inputs, that the reading file information is obtained correctly and the computation of the hypercyle, index and admin-control-list-length is faultless.

6.2 Configuration file (XML file)

As we have already explained, to configure the switches we need to provide the *configuration file*,which is a XML file that corresponds to the YANG model. To generate the *configuration file* we have followed the XML mapping rules described in the document RFC6020. The XML format generated from our YANG model is depicted in Listing 6.1 .

Notice in the leaf <interface> the type should be ethernetCsmacd but in the real XML file it will be represented with the numbers "1234" because the platform where the project is based does not supports strings yet.
The XML file contains the data that will be send from the client to the server for modifying the configuration. To fill all the fields of the XML file we use the configuration parameters that has been determined in the section 6.1. Even though we have individually testes this mechanism we have decided to only attach in the Appendix A the XML files generated during the validation phase. This decision has been taken in order to do not extend the document more.

```
<ietf-interfaces-interfaces xmlns="urn:Switch:test">
  <interface>
    <name>switch-number1</name>
    <type>EthernetCsmacd</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-shed-gate-parameters>
        <admin-cycle-time>
          <denominator>500/hypercycle</denominator>
        </admin-cycle-list>
        <config-change>true</config-change>
        <admin-control-list>
          <index>n</index>
          <sgs-params>
            <time-interval-value>period
            </time-interval-value>
            <gates-states-value>gates-state
            </gates-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>n-1</index>
```

```
          <sgs-params>
            <time-interval-value>period
            </time-interval-value>
            <gates-states-value>gates-state
            </gates-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>hypercycle
        </admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
  <interface>
    <name>switch-number2</name>
      . . .
  </interface>
    . . .
</ietf-interfaces-interfaces>
```

Listing 6.1: Format of the configuration file

6.3 Implementation of the user database

In the section 5.3 we decided to use a MySQL as a database management system. Using MySQL we have implemented the *user database* designed in Section 4.6. Even though MySQL is a relational database, we have decided to only create one entity. Previously to take this decision we tried to implement a database with various entities but when we tried to create the relations between them, using the provided functions, we found several problems. Since the main objective of the project is not to create a database we decided to use only one entity called gate control list.

The entity gate control list includes all the configuration parameters decided in the design (see section 4.4). The objective of the *user database* is to store the different configurations and, thus, we have created the attribute list-number. So for each list-number can exist a gate control list for each port of each switch. Also we have to consider that for different list-numbers could exist a gate control list for the same port of the same switches. Having that considerations in mind we have decided to use as a primary key all the attributes except the list-number.

To manage the database we have the library MySQL (see section 5.3). Using MySQL we can make queries that allows us to get, modify or delete information of the *user database*. Since we are going to add and delete gate control lists from the database we need queries. The cases when we require to used queries and its specific query are as follows:

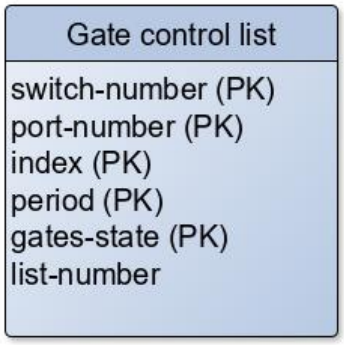


Figure 6.1: *Entity*

– To know the corresponding value of list-number, we need to know the value of the highest list-number that already exists in the database in order to not overwrite the previously stored configurations. Therefore the new configuration will be stored and the list-number corresponding to it will be the current highest list-number plus one. Then we need to retrieve the highest list-number from the switch whose gate control lists we want to store. For retrieving the highest list-number of a switch-number from the database, we have make a query that orders the list-number from the highest value to the lowest value and then only read the first row that gives us the highest list-number. The query that has been used is shown next. The variable Aswitch-number makes reference to the switch-number of the current switch's gate control lists that want to be stored:

> "SELECT list-number, switch-number FROM gate-control-list GROUP BY switch-number, list-number HAVING switch-number = Aswitch-number ORDER BY list-number DESC"

– To store a new configuration, we have to introduce all the parameters in the database at once because the attributes are defined as primary keys. Therefore they can not have the value null, and the list-number has been defined as no null. The query to performance the storage is showed below where the suffix value is used to indicate the variables that are being stored :

> "INSERT INTO gate-control-list(switch-number, port-number, list-number, index, period, gates-state) VALUES (switch-number-value, port-number-value, list-number-value, period-value, gates-state-value)"

– To set a configuration that is in the database we need to retrieve the port-numbers and its corresponding gate control list—formed by a list of period and gates-state—. The gate control list must be retrieved keeping the order indicated by the index. To obtain this parameters we use the following query:

"SELECT port-number, period, gates-state FROM gate-control-list WHERE switch-number = Bswitch-number AND list-number = Alist-number ORDER BY port-number, index ASC"

– The user can see a list with the attributes list-number and switch-number that gives information about which list-numbers are available for each switch (1). The user also can see the configuration corresponding to one list-number and switch-number (2), it means that given those two parameters he would get the corresponding gate control lists of each port. Below there are the queries that perform this tasks respectively:

(1) "SELECT DISTINCT list-number, switch-number FROM INFO ORDER BY switch-number"
(2) "SELECT port-number, index, period, gates-state FROM INFO WHERE switch-number = Cswitch-number ORDER BY port-number, index ASC"

– Finally the user has three options for removing configurations from the database. The first option (1) consists in indicating the switch-number and list-number to remove that configuration The second (2) possibility is delete all the configurations from one switch indicating the switch-number. The last option (3) is delete all the configurations from the database. These options are respectively represented in the following queries:

(1) "DELETE FROM gates-control-list WHERE switch-number = Dswitch-number AND list-number = Blist-number"
(2) "DELETE FROM gate-control-list WHERE switch-number = switch-number-value"
(3) "DELETE FROM gate-control-list"

Notice that there might be some nomenclature differences between these queries and the ones in the program.

6.4 Implementation of the client

For the implementation of the client we have taken, as starting point, an already existing client from the library Netopeer2. The Netopeer2 client (see section 5.2) gets the commands from the user trough the console. In order to include the client in our implementation we have to modify the code to perform the connection with the server—that is in the switches— and the NETCONF operations to configure the switches. To set the configuration the <copy-config> operation with the running datastore as a target would be enough, but only using the <copy-config> interferences could happen. In order to avoid any possible interference we have decided to use the <lock> operation to avoid other sessions modify the datstore. In a real system probably several modifications

would be done in the configuration before it is set in the running datastore; for this reason we have opted to modify the candidate datastore and then use the <commit> operation to set the configuration on the running datastore. Once the operation has been carried out we unblock the datastore using the <unlock> operation. The sequence of NETCONF operations to be executed can be seen below:

```
<lock (target = running)>
<lock (target = candidate)>
<copy-config (source = XML file, target = candidate)>
<commit>
<unlock (target = running)>
<unlock (target = candidate)>
```

Considering that we have two switches and the network could be extended, the sequence of operations will be repeated for each switch every time a configuration is going to be set. Besides, the client must have access to the each address of the switches and the channel that will be used to create the SSH connection. In our case the switches IP are 192.168.4.64 and 192.168.4.65. For the SSH communication both the server and client are configured to use the port 830. We must remember that the client includes the previous implementations; to join the program of the client and the one corresponding to the process of the configuration parameters and the management of the database we have used a simple bash script.

CHAPTER 7

INTEGRATION AND VALIDATION

In this chapter it is explained how the implemented mechanisms have been integrated to obtain the whole automatic configuration system. The integration has consisted in uploading the server, which belongs to the library Netopeer2, in the switches and our client in the PC. The first approach to validate our implementation has consisted in trying the system in the PC. We have decided to do this first validation because it ensures that there will not be communication problems that could occur the switches. Once done this validation we have proceeded with the validation using both PC and switches. Additionally, to validate that the configuration is set correctly in the server-side we have included the <get-config> operation, indicating the running datastore as the target, after the <commit> so the server will return the data letting us check that it has been correctly modified. All the results of the tests can be found in the Appendix A.

7.1 Validation on PC

For accomplishing this test we have run both the server and the client in the PC, this is what we call the virtualized network. To carry out the validation on the PC (virtualized network), the program corresponding to the client has been modified so it will work as if it was the switch whose IP is 192.168.4.64.

For the first test we have used a reading file with only one switch configuration and we have directly used the reading file to set the configuration. The results of this test has been positive and therefore we have proceeded with the following tests.

The second test has been done in order to ensure that a configuration can be set from the database. For this purpose, we have created a reading file with multiple switches and then we stored the configurations in the *user database*. Having the configurations in the *user database* we have used the commands to visualise the configurations and check that there is not any mistake. After that, we have selected one configuration from the *user database* to be uploaded and the configuration have been set. To ensure

that the the configurations can be deleted, we have delete the configurations previously stored and then we have shown the available configurations confirming that the delete functionality works.

7.2 Validation on switches

From the validation on the PC, we extracted that the communication server-client and the database work correctly and hence we could continue performing tests. To validate the implementation on the switches (real network) we have connected the two switches between them with an Ethernet wire and then one of them to the PC. The reason to have chosen this connections is because it requires less wires than connecting both switches to the PC. Since only one switch is connected to the PC, this switch has the function of retransmitting the messages exchanged between the PC and the other switch. For the tests the switch connected to the PC is the one with the IP 192.168.4.65, and consequently the switch whose IP is 192.168.4.65 is only connected to the other switch. This set up is shown in the figure

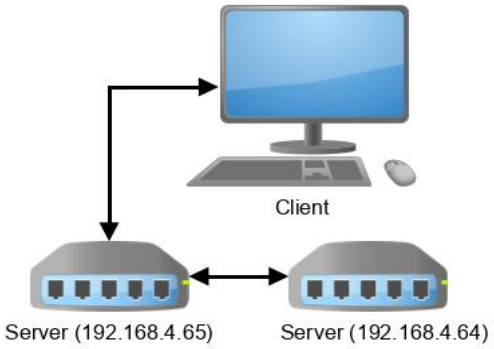


Figure 7.1: Connections used for the validation

Before trying to configure both switches at the same time, we have tested the switch that is directly connected to the PC. The test consisted in storing two configurations into the database and then upload them into the switch one after the other, verifying that all the parameters have been faultlessly set.

The next test was done to ensure that the switch that is not directly connected to the PC can be configured. Once this test has been run successfully we can ensure that both datastores inside the switches can be individually configured.

The final test is a modification of both switches datastores in the same execution. This is done applying the configuration directly from the configuration file. The positive result of the test concluded that the mechanism works as it was expected.

We must take into account that even the validation has been done only using two switches— because there were not more of them available— the mechanism will work for a network with N switches. The reason for this, it is that the routing of the switches is not seen by NETCONF and it only depends on the IP of the switches.

CHAPTER 8

CONCLUSIONS

This last chapter summarises the tasks accomplished during the project and the conclusions drawn from the results. In addition, we propose some ideas for future work based on this project.

8.1 Summary

The main objective of this project was to design, implement and validate a system that provides a solution to automatically configure the TAS parameters in TSN switches. The TAS is the set of traffic shapers that rule the communication so as to guarantee the QoS and the real-time requirements of the different classes of traffic.

To carry out the design and the implementation, I had to learn about TSN. The most relevant TSN standard have been the IEEE 802.1Qbv, which describes the TAS mechanism. Additionally, I learned more about TSN in order to design an automatic configuration system that is compliant with the TSN standards and, thus, it can be deployed on a TSN network. Besides, I investigated about NETCONF— which is the network management protocol that is used— and YANG that is the data model language to be used by NETCONF. This part of the work is represented by Chapter 2.

Once I knew about the technologies related to the project, I designed the architecture and the mechanisms of the automatic configuration system. Concerning the architecture, we took as reference the IEEE Std 802.1Qcc, which describes three architecture models from which we chose the *Fully centralised user model*. Since we wanted to use NETCONF for configuring the network, the architecture is a client/server one where the CNC of the *Fully centralised user model* is the client (PC) and each switch is a server. As concerns the design tasks themselves, we first had to design the procedure to set a new configuration, which is described in Section 4.3. Then we designed the reading file that allows the user to introduce the configuration that he wants to set on the switch. In addition, we specified the YANG data model that organises the parameters of the TAS in the *TAS database* of our switches. Apart from that, the candidate decided to

introduce the *user database* on the clien-side, so that the user does not need to store all the configuration in just files. By means of these tasks, we have accomplished the entire design of the automatic configuration system of the TAS, whose specifications are in Chapter 4.

Since there was not an already-available platform to implement our configuration system, we investigated what tools were more suitable for doing so. Basically, the tools had to provide us the facilities to work with NETCONF and YANG. Initially, the research lead us towards the library Libnetconf2, but while trying to perform the implementation using Libnetconf2 we found Netopeer2. Netopeer2 includes Libnetconf2 and gives a certain level of abstraction that allows us to work with NETCONF without having to manage the SSH session. Regarding to the database, the candidate decided to use MySQL, based on his experience in a subject during the degree. These tools are explained in the Chapter 5.

Regarding to the implementation, which is described in Chapter 6, it was split in different parts to reduce its complexity. First I implemented the mechanism to extract the values of the configuration parameters from the reading file provided by the user. Then, using the previous values we generated the *configuration file*, which is the XML file to be used to send the configuration data from the client (PC) to the server (switch). Afterwards, I implemented the *user database* to store multiple configurations. The final part of the implementation consisted in modifying the client, provided by the library Netopeer2, in order to execute the NETCONF operations to modify the configuration in the *TAS database* of the servers (switches).

Finally, in Chapter 7 I integrated the implemented mechanisms so as to obtain the whole reconfiguration system and, then, I carried out a series of tests to ensure the correctness of the system. The integration consisted in uploading the mechanisms to the client(PC) and the server (switch) respectively. Concerning the tests, I first carried out the test on a virtualised network and, then, on a real one. The tests on virtualized network ensure that the mechanisms work as intended, and detect and fix possible connections errors; whereas the tests on the real network verify the correct performance of the whole system in a real TSN prototype.

We conclude that the objectives proposed for this Bachelor Thesis have been achieved. Additionally, we believe that our automatic configuration system can be applied to TSN switches other than the ones we have used, by using the YANG data model of those other switches.

8.2 Future work

The objectives of the project have been achieved and the implementation that has been carried out in this project can be used as a starting point for future projects.

The accomplished automatic configuration system of the TAS gets the values of the configuration parameters from a user; but instead of a user there could be an scheduler that computes the those values based on the requirements of the network. So in this sense the combination of the scheduler and the implemented system would constitute the CNC. The combination of the automatic configuration system and an scheduler would allow to create an autonomous system that calculates the configura-

tion parameters, from the requirements of the network, and then it sets them on the switches.

This project have been carried out on the control plane. For this reason the configurations are set on the *TAS database* of the switch but this configuration is not applied to forward/transmit the traffic. This bachelor thesis has not included the data plane—the part in charge of actually applying the configuration to forward/transmit the traffic—because the vendor is not willing to share the required information to make it. For this reason, the data plane implementation would be a necessary work for including the switches in a real system.

CHAPTER 9

APPENDIX A

This chapter contains the results from the different tests that have been done in Chapter 7.Since the part of the program in charge of generating the *configuration file* does not depend whether the test in carried out on the PC or on the switches, we only include the *configuration file* for some tests but not for all of them.

9.1 Tests on PC

9.1.1 Test 1

```
switch: 64
    port_number: 1
        200000,10000000
        100000,01000000
        100000,00100000
        100000,00010000
        100000,00001000
        100000,00000100
        100000,00000010
        100000,00000001
```

Figure 9.1: Reading file used in the first test on the PC

```
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>555555.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>32</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>3</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>16</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>4</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>8</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>5</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>4</gate-states-value>
          </sgs-params>
        </admin-control-list>
```

Figure 9.2: First test on PC part 1/2

```
        <admin-control-list>
          <index>6</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>2</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>7</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>1</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>8</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.3: First test on PC part 2/2

```
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
    <interface>
        <name>1</name>
        <type>1234</type>
        <enabled>true</enabled>
        <ieee802-dot1q-bridge-bridge-port>
            <ieee802-dot1q-sched-gate-parameters>
                <admin-cycle-time>
                    <denominator>555555</denominator>
                    <numerator>500</numerator>
                </admin-cycle-time>
                <config-change>true</config-change>
                <admin-control-list>
                    <index>0</index>
                    <sgs-params>
                        <time-interval-value>200000</time-interval-value>
                        <gate-states-value>128</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list>
                    <index>1</index>
                    <sgs-params>
                        <time-interval-value>100000</time-interval-value>
                        <gate-states-value>64</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list>
                    <index>2</index>
                    <sgs-params>
                        <time-interval-value>100000</time-interval-value>
                        <gate-states-value>32</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list>
                    <index>3</index>
                    <sgs-params>
                        <time-interval-value>100000</time-interval-value>
                        <gate-states-value>16</gate-states-value>
                    </sgs-params>
                </admin-control-list>
```

Figure 9.4: *Configuration file* of the first test on PC, part 1/2

```
41                  <admin-control-list>
42                      <index>4</index>
43                      <sgs-params>
44                          <time-interval-value>100000</time-interval-value>
45                          <gate-states-value>8</gate-states-value>
46                      </sgs-params>
47                  </admin-control-list>
48                  <admin-control-list>
49                      <index>5</index>
50                      <sgs-params>
51                          <time-interval-value>100000</time-interval-value>
52                          <gate-states-value>4</gate-states-value>
53                      </sgs-params>
54                  </admin-control-list>
55                  <admin-control-list>
56                      <index>6</index>
57                      <sgs-params>
58                          <time-interval-value>100000</time-interval-value>
59                          <gate-states-value>2</gate-states-value>
60                      </sgs-params>
61                  </admin-control-list>
62                  <admin-control-list>
63                      <index>7</index>
64                      <sgs-params>
65                          <time-interval-value>100000</time-interval-value>
66                          <gate-states-value>1</gate-states-value>
67                      </sgs-params>
68                  </admin-control-list>
69                  <admin-control-list-length>8</admin-control-list-length>
70                  <admin-base-time>
71                      <seconds>0</seconds>
72                      <fractional-seconds>0</fractional-seconds>
73                  </admin-base-time>
74              </ieee802-dot1q-sched-gate-parameters>
75          </ieee802-dot1q-bridge-bridge-port>
76      </interface>
77  </ietf-interfaces-interfaces>
78
```

Figure 9.5: *Configuration file* of the first test on PC, part 2/2

9.1.2 Test 2

```
switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000
          100000,00010000
          100000,00001000
          100000,00000100
          100000,00000010
          100000,00000001

switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000

     port_number: 2
          300000,10000001
          600000,00000011
          300000,00000011
     port_number: 3
          800000,10000011
          400000,10000010
          600000,10000001
switch: 65
     port_number: 1
          30,00000011
          40,00001100
          50,00001110
```

Figure 9.6: Reading file used in the second test on the PC

```
help
store                       ->store the new lists in the database
store-upload                ->store the new lists and upload the configuration
upload                      ->upload the configuration from the file
upload-config               ->upload the chosen configuration from the database
choose-config --switch [switch ID] --list [list ID]   ->choose the configuration to be uploaded
select                      ->Show the lists of each switch
select --switch [switch ID] --list [list ID]  ->show the chosen list of a switch
delete                      ->remove all the infomation in the database
delete --switch [switch ID] ->remove the all the lists from the switch
delete --switch [switch ID] --list [list ID] ->remove the desired list from the database
select
   List ID    | Switch number
--------------|------------
store
select
   List ID    | Switch number
--------------|------------
      1           64
      2           64
      1           65
select --switch 64 --list 2
Port number | Index |    Period    | Gates state
------------|-------|--------------|------------
    1         0        200000           128
    1         1        100000           64
    1         2        100000           32
    2         0        300000           129
    2         1        600000           3
    2         2        300000           3
    3         0        800000           131
    3         1        400000           130
    3         2        600000           129
choose-config --switch 64 --list 2
upload-config
```

Figure 9.7: Second test on PC part 1/4

```
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>1250000.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>32</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
  <interface>
    <name>2</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>416666.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>300000</time-interval-value>
            <gate-states-value>129</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
```

Figure 9.8: Second test on PC part 2/4

```
          <index>2</index>
          <sgs-params>
            <time-interval-value>300000</time-interval-value>
            <gate-states-value>3</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
  <interface>
    <name>3</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>277777.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>800000</time-interval-value>
            <gate-states-value>131</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>400000</time-interval-value>
            <gate-states-value>130</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>600000</time-interval-value>
            <gate-states-value>129</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.9: Second test on PC part 3/4

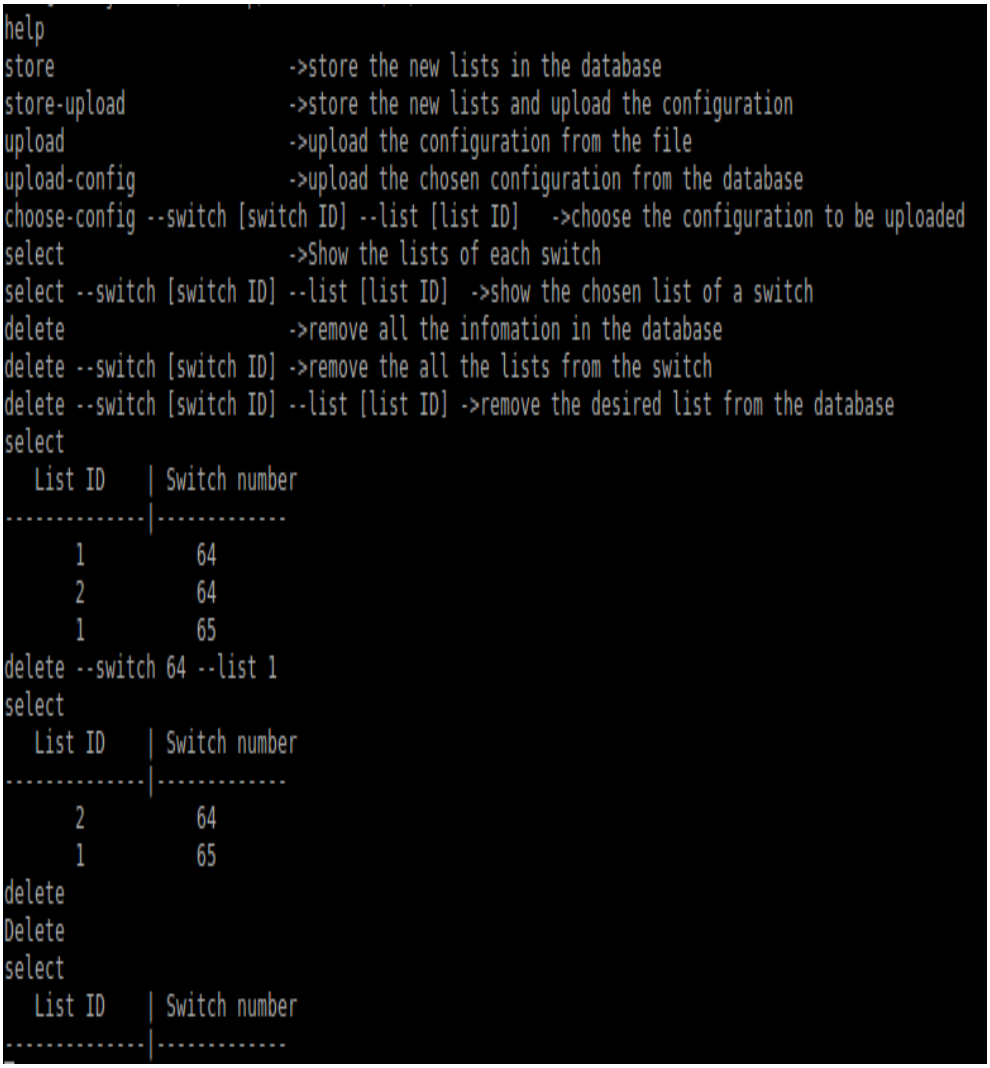


Figure 9.10: Second test on PC part 4/4

```
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
    <interface>
        <name>1</name>
        <type>1234</type>
        <enabled>true</enabled>
        <ieee802-dot1q-bridge-bridge-port>
            <ieee802-dot1q-sched-gate-parameters>
                <admin-cycle-time>
                    <denominator>1250000</denominator>
                    <numerator>500</numerator>
                </admin-cycle-time>
                <config-change>true</config-change>
                <admin-control-list>
                    <index>0</index>
                    <sgs-params>
                        <time-interval-value>200000</time-interval-value>
                        <gate-states-value>128</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list>
                    <index>1</index>
                    <sgs-params>
                        <time-interval-value>100000</time-interval-value>
                        <gate-states-value>64</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list>
                    <index>2</index>
                    <sgs-params>
                        <time-interval-value>100000</time-interval-value>
                        <gate-states-value>32</gate-states-value>
                    </sgs-params>
                </admin-control-list>
                <admin-control-list-length>3</admin-control-list-length>
                <admin-base-time>
                    <seconds>0</seconds>
                    <fractional-seconds>0</fractional-seconds>
                </admin-base-time>
            </ieee802-dot1q-sched-gate-parameters>
        </ieee802-dot1q-bridge-bridge-port>
```

Figure 9.11: *Configuration file* of the second test on PC, part 1/3

```
41      </interface>
42      <interface>
43          <name>2</name>
44          <type>1234</type>
45          <enabled>true</enabled>
46          <ieee802-dot1q-bridge-bridge-port>
47              <ieee802-dot1q-sched-gate-parameters>
48                  <admin-cycle-time>
49                      <denominator>416666</denominator>
50                      <numerator>500</numerator>
51                  </admin-cycle-time>
52                  <config-change>true</config-change>
53                  <admin-control-list>
54                      <index>0</index>
55                      <sgs-params>
56                          <time-interval-value>300000</time-interval-value>
57                          <gate-states-value>129</gate-states-value>
58                      </sgs-params>
59                  </admin-control-list>
60                  <admin-control-list>
61                      <index>1</index>
62                      <sgs-params>
63                          <time-interval-value>600000</time-interval-value>
64                          <gate-states-value>3</gate-states-value>
65                      </sgs-params>
66                  </admin-control-list>
67                  <admin-control-list>
68                      <index>2</index>
69                      <sgs-params>
70                          <time-interval-value>300000</time-interval-value>
71                          <gate-states-value>3</gate-states-value>
72                      </sgs-params>
73                  </admin-control-list>
74                  <admin-control-list-length>3</admin-control-list-length>
75                  <admin-base-time>
76                      <seconds>0</seconds>
77                      <fractional-seconds>0</fractional-seconds>
78                  </admin-base-time>
79              </ieee802-dot1q-sched-gate-parameters>
80          </ieee802-dot1q-bridge-bridge-port>
```

Figure 9.12: *Configuration file* of the second test on PC, part 2/3

```
81      </interface>
82      <interface>
83          <name>3</name>
84          <type>1234</type>
85          <enabled>true</enabled>
86          <ieee802-dot1q-bridge-bridge-port>
87              <ieee802-dot1q-sched-gate-parameters>
88                  <admin-cycle-time>
89                      <denominator>277777</denominator>
90                      <numerator>500</numerator>
91                  </admin-cycle-time>
92                  <config-change>true</config-change>
93                  <admin-control-list>
94                      <index>0</index>
95                      <sgs-params>
96                          <time-interval-value>800000</time-interval-value>
97                          <gate-states-value>131</gate-states-value>
98                      </sgs-params>
99                  </admin-control-list>
100                 <admin-control-list>
101                     <index>1</index>
102                     <sgs-params>
103                         <time-interval-value>400000</time-interval-value>
104                         <gate-states-value>130</gate-states-value>
105                     </sgs-params>
106                 </admin-control-list>
107                 <admin-control-list>
108                     <index>2</index>
109                     <sgs-params>
110                         <time-interval-value>600000</time-interval-value>
111                         <gate-states-value>129</gate-states-value>
112                     </sgs-params>
113                 </admin-control-list>
114                 <admin-control-list-length>3</admin-control-list-length>
115                 <admin-base-time>
116                     <seconds>0</seconds>
117                     <fractional-seconds>0</fractional-seconds>
118                 </admin-base-time>
119             </ieee802-dot1q-sched-gate-parameters>
120         </ieee802-dot1q-bridge-bridge-port>
121     </interface>
122 </ietf-interfaces-interfaces>
```

Figure 9.13: *Configuration file* of the second test on PC, part 3/3

9.2 Tests on switches

9.2.1 Test 1

```
switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000
          100000,00010000
          100000,00001000
          100000,00000100
          100000,00000010
          100000,00000001

switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000
          300000,10000001
          600000,00000011

     port_number: 2
          300000,10000001
          600000,00000011
          300000,00000011
     port_number: 3
          800000,10000011
          400000,10000010
          600000,10000001
switch: 65
     port_number: 1
          30,00000011
          300000,10000001
          600000,00000011
          50,00001110
     port_number: 3
          200000,10000000
          100000,01000000
          100000,00100000
switch: 65
     port_number: 1
          100000,00010000
          100000,00001000
          50,00001110
```

Figure 9.14: Reading file used in the test on switch with IP 192.168.4.65

```
select
   List ID    | Switch number
--------------|-------------
       1           64
       2           64
       1           65
       2           65
choose-config --switch 65 --list 1
upload-config
Interactive SSH Authentication
Type your password:
Password:
OK
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>555506.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>30</time-interval-value>
            <gate-states-value>3</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>300000</time-interval-value>
            <gate-states-value>129</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>600000</time-interval-value>
            <gate-states-value>3</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>3</index>
          <sgs-params>
            <time-interval-value>50</time-interval-value>
            <gate-states-value>14</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>4</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
```

Figure 9.15: First test on switch 192.168.4.65 part 1/2

```
  <interface>
    <name>3</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>1250000.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>32</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.16: First test on switch 192.168.4.65 part 2/2

9.2.2 Test 2

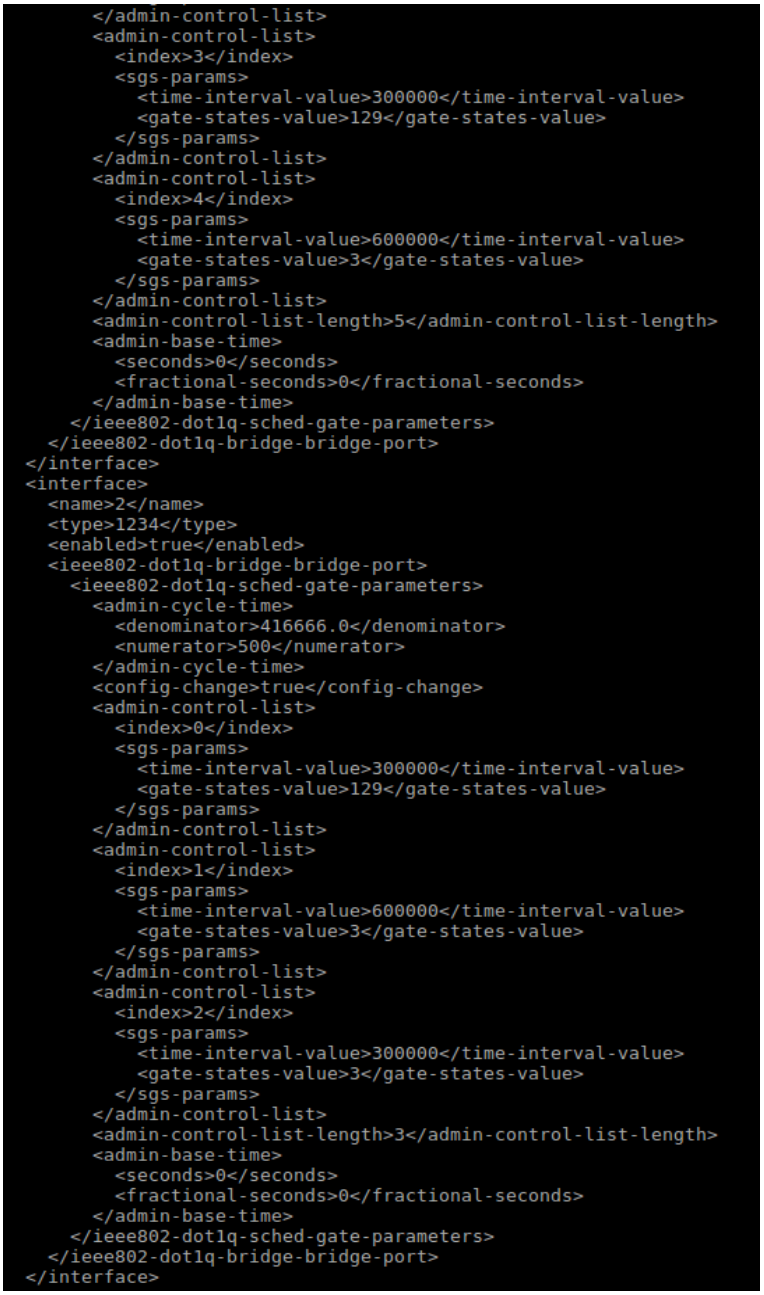
```
select
   List ID    | Switch number
--------------|-------------
       1           64
       2           64
       1           65
       2           65
choose-config --switch 65 --list 2
upload-config
Interactive SSH Authentication
Type your password:
Password:
OK
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>2499375.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>16</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>8</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>50</time-interval-value>
            <gate-states-value>14</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.17: Second test on switch 192.168.4.65 part 1/1

9.2.3 Test 3

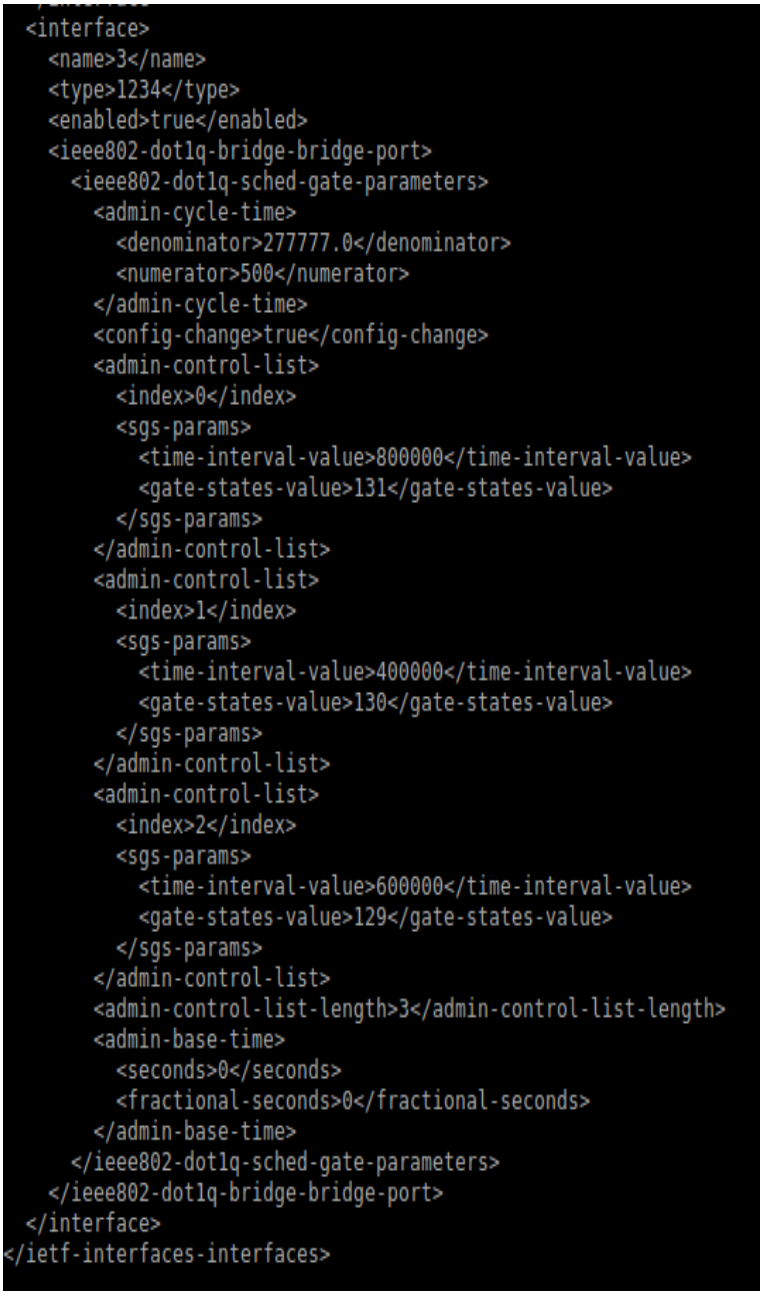
```
select
   List ID    | Switch number
--------------|-------------
store
select
   List ID    | Switch number
--------------|-------------
       1           64
       2           64
       3           64
       1           65
       2           65
       3           65
select --switch 64 --list 1
Port number | Index |    Period    | Gates state
------------|-------|--------------|-------------
    1         0        200000           128
    1         1        100000           64
    1         2        100000           32
    1         3        300000           129
    1         4        600000           3
    2         0        300000           129
    2         1        600000           3
    2         2        300000           3
    3         0        800000           131
    3         1        400000           130
    3         2        600000           129
choose-config --switch 64 --list 1
upload-config
Interactive SSH Authentication
Type your password:
Password:
OK
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>384615.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>32</gate-states-value>
          </sgs-params>
```

Figure 9.18: First test on switch 192.168.4.64 part 1/3



```
      </admin-control-list>
      <admin-control-list>
        <index>3</index>
        <sgs-params>
          <time-interval-value>300000</time-interval-value>
          <gate-states-value>129</gate-states-value>
        </sgs-params>
      </admin-control-list>
      <admin-control-list>
        <index>4</index>
        <sgs-params>
          <time-interval-value>600000</time-interval-value>
          <gate-states-value>3</gate-states-value>
        </sgs-params>
      </admin-control-list>
      <admin-control-list-length>5</admin-control-list-length>
      <admin-base-time>
        <seconds>0</seconds>
        <fractional-seconds>0</fractional-seconds>
      </admin-base-time>
    </ieee802-dot1q-sched-gate-parameters>
  </ieee802-dot1q-bridge-bridge-port>
</interface>
<interface>
  <name>2</name>
  <type>1234</type>
  <enabled>true</enabled>
  <ieee802-dot1q-bridge-bridge-port>
    <ieee802-dot1q-sched-gate-parameters>
      <admin-cycle-time>
        <denominator>416666.0</denominator>
        <numerator>500</numerator>
      </admin-cycle-time>
      <config-change>true</config-change>
      <admin-control-list>
        <index>0</index>
        <sgs-params>
          <time-interval-value>300000</time-interval-value>
          <gate-states-value>129</gate-states-value>
        </sgs-params>
      </admin-control-list>
      <admin-control-list>
        <index>1</index>
        <sgs-params>
          <time-interval-value>600000</time-interval-value>
          <gate-states-value>3</gate-states-value>
        </sgs-params>
      </admin-control-list>
      <admin-control-list>
        <index>2</index>
        <sgs-params>
          <time-interval-value>300000</time-interval-value>
          <gate-states-value>3</gate-states-value>
        </sgs-params>
      </admin-control-list>
      <admin-control-list-length>3</admin-control-list-length>
      <admin-base-time>
        <seconds>0</seconds>
        <fractional-seconds>0</fractional-seconds>
      </admin-base-time>
    </ieee802-dot1q-sched-gate-parameters>
  </ieee802-dot1q-bridge-bridge-port>
</interface>
```


Figure 9.19: First test on switch 192.168.4.64 part 2/3



```
  <interface>
    <name>3</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>277777.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>800000</time-interval-value>
            <gate-states-value>131</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>400000</time-interval-value>
            <gate-states-value>130</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>2</index>
          <sgs-params>
            <time-interval-value>600000</time-interval-value>
            <gate-states-value>129</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>3</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.20: First test on switch 192.168.4.64 part 3/3

9.2.4 Test 4

```
switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000
          300000,10000001
          600000,00000011

     port_number: 2
          300000,10000001
          600000,00000011
          300000,00000011
     port_number: 3
          800000,10000011
          400000,10000010
          600000,10000001
switch: 64
     port_number: 1
          200000,10000000
          100000,01000000
          100000,00100000
          100000,00010000
          100000,00001000
          100000,00000100
          100000,00000010
          100000,00000001
switch: 64
     port_number: 1
          200000,10000000
          100000,01000000

switch: 65
     port_number: 1
          300000,00000011
          300000,10000001
          600000,00000011
          500000,00001110
     port_number: 3
          200000,10000000
          100000,01000000
          100000,00100000
switch: 65
     port_number: 1
          100000,00010000
          100000,00001000
          50,00001110
switch: 65
     port_number: 1
          200000,10000000
          100000,01000000
```

Figure 9.21: Reading file used in the test on both switchwa

```
upload
Interactive SSH Authentication
Type your password:
Password:
OK
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>1666666.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>2</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>

Interactive SSH Authentication
Type your password:
Password:
OK
DATA
<ietf-interfaces-interfaces xmlns="urn:NewSwitch:test">
  <interface>
    <name>1</name>
    <type>1234</type>
    <enabled>true</enabled>
    <ieee802-dot1q-bridge-bridge-port>
      <ieee802-dot1q-sched-gate-parameters>
        <admin-cycle-time>
          <denominator>1666666.0</denominator>
          <numerator>500</numerator>
        </admin-cycle-time>
        <config-change>true</config-change>
        <admin-control-list>
          <index>0</index>
          <sgs-params>
            <time-interval-value>200000</time-interval-value>
            <gate-states-value>128</gate-states-value>
          </sgs-params>
        </admin-control-list>
```

Figure 9.22: First test on both switches part 1/2

```
        </admin-control-list>
        <admin-control-list>
          <index>1</index>
          <sgs-params>
            <time-interval-value>100000</time-interval-value>
            <gate-states-value>64</gate-states-value>
          </sgs-params>
        </admin-control-list>
        <admin-control-list-length>2</admin-control-list-length>
        <admin-base-time>
          <seconds>0</seconds>
          <fractional-seconds>0</fractional-seconds>
        </admin-base-time>
      </ieee802-dot1q-sched-gate-parameters>
    </ieee802-dot1q-bridge-bridge-port>
  </interface>
</ietf-interfaces-interfaces>
```

Figure 9.23: First test both switches part 2/2

BIBLIOGRAPHY

[1] D. Gessner, "Adding fault tolerance to a flexiblereal-time ethernet network for embedded systems," pp. 101–111, 2017. (document), 2.1

[2] V. Gruzauskas and V. Navickas, "Cyber?physical systems expression in industry 4.0 context," *Financial and credit activity: problems of theory and practice*, vol. 2, 02 2018. (document), 2.2

[3] y. H. Lim, M. Waltl, F. Chaari and D. Herrscher, "Performance analysis of the ieee 802.1 ethernet audio/video bridging standard." (document), 2.3

[4] "Ieee standard for local and metropolitan area networks?bridges and bridged networks amendment 31: Stream reservation protocol (srp) enhancements and performance improvements," 2018. (document), 1.3, 2.4, 2.5, 2.6

[5] "Ieee standard for local and metropolitan area networks?bridges and bridged networks amendment 25: Enhancements for scheduled traffic," 2015. (document), 2.7, 2.8, 2.9

[6] N. Cognoms-autor, *MTSN Kit User Guide*. Editor, 2018. (document), 4.1, 4.2, 4.3

[7] H. Kopetz, "Design principles for distributed embedded applications," *Kluwer Academic Publishers*, 1997. 1.1

[8] W. Steiner, "Next generation real-time networks based on it technologies," *In proceedings of the 2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1 –8, 2016. 1.1

[9] "TTTech," https://www.tttech.com/. 1.1

[10] A. Burns, *Real-Time Systems And Programming Languages: Ada, Real-Time Java, And C/Real-Time POSIX*. 1.2

[11] "Yang - a data modeling language for the network configuration protocol (netconf)," Internet Engineering Task Force (IETF), Standard RFC 6020, 2010. 1.3, 2.3

[12] "Network configuration protocol (netconf)," Internet Engineering Task Force (IETF), Standard RFC 6241, 2011. 1.3

[13] S. S. B. Guarav, B. Saurabh and M. Arsalan, "Osi reference model: An overview," *International Journal of Computer Trends and Technology (IJCTT)*, 2014. 2.1.3