Dimensioning TSN network synchronization in different embedded contexts

Quentin Bailleul

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *24 novembre 2023* **par :**

Quentin BAILLEUL

Dimensionnement de la synchronisation d'un réseau TSN dans différents contextes embarqués

---

JURY

Ye-Qiong Song	Professeur à l'Université de Lorraine	Rapporteur
Nicolas Navet	Professeur à l'Université du Luxembourg	Rapporteur
Thomas Watteyne	Directeur de recherches INRIA	Membre du Jury
Marc Boyer	Directeur de recherches ONERA	Membre du Jury
Raphaël Naves	Architecte réseau Dassault Aviation	Membre du Jury
Katia Jaffrès-Runser	Professeur à Toulouse INP	Co-directrice de Thèse
Jean-Luc Scharbarg	Professeur à Toulouse INP	Co-directeur de Thèse
Philippe Cuenot	Architecte logiciel Continental	Encadrant industriel

---

École doctorale et spécialité :
MITT : Domaine STIC : Réseaux, Télécoms, Systèmes et Architecture

Unité de Recherche :
IRIT - RMESS

Directeur(s) de Thèse :
Katia JAFFRES-RUNSER, *Jean-Luc SCHARBARG* et *Phillipe CUENOT*

Rapporteurs :
Ye-Qiong Song et *Nicolas Navet*

Abstract

Distributed critical embedded systems faced a growing need for performance, which cannot be achieved without improving the communication architecture. To cope with this need, IEEE 802.1 Time-Sensitive Networking (TSN) working group proposes a set of standards known as the TSN standard to extend Ethernet. These new protocols quickly gained the interest of the embedded industrial world by the promise of a unified network adaptable to numerous data rates with numerous Quality Of Service (QoS) mechanisms intended to offer different levels of determinism. Among the new mechanisms proposed, some mechanisms, such as the Time-Aware Shaper or the Cyclic Queuing and Forwarding, stand out due to the need for a common time base throughout the entire network in order to function optimally. This network-wide synchronization is provided by the IEEE 802.1AS standard which is the adaptation for TSN networks of IEEE 1588, better known under the name Precision Time Protocol (PTP).

The deployment of TSN as the backbone of an embedded network in the context of automotive, aeronautical or space worlds is a substantial novelty. Time-triggered shapers are of particular interest to offer a jitter-less communication delay to very critical flows. IEEE802.1AS becomes thus a de-facto service that has to be rolled out to provide network-wide synchronisation. Since critical communications require determinism and robustness, the underlying synchronisation service has to offer the same level of service garanties. This thesis studies the design and deployment of IEEE 802.1AS to offer a robust and precise network-wide synchronisation.

To do so, we question first the achievable synchronization precision of IEEE802.1AS. We start by presenting a set of improvements made to an open source simulator of the protocol, carried out with the aim of making it representative of reality. Then, based on the knowledge acquired with the simulator, a formal model allowing the calculation of a bound on the worst case precision is derived. These two contributions are assessed by intensive measurements obtained on hardware supporting IEEE 802.1AS and it allows us to study the parameters impacting the synchronization precision.

Secondly, we question the ability of the protocol to withstand link and device failures. Thus, a comparison between the two robustness mechanisms proposed by the standard is made. This study highlights that the static robustness mechanism better meets the needs of critical environments at the cost of a more complex manual configuration. In order to facilitate the design of a static robust synchronization spanning tree in the network, we propose a practical methodology that looks for the most precise synchronization tree among the most robust ones.

And finally, we put into practice, on the case study of a satellite embedded network, the results obtained previously in order to study the impact of the precision and the robustness on the rest of the network activities. We show that the proposed IEEE 802.1AS design offers the required precision guaranty using a very small fraction of the network bandwidth and with a limited impact on the timeliness of lower priority flows.

Résumé Français

Les systèmes distribués embarqués critiques sont aujourd'hui face à un besoin grandissant de performance, qui ne pourra pas être atteint sans amélioration de l'architecture de communication. C'est pour répondre à ce besoin que le groupe de travail IEEE 802.1 Time-Sensitive Networking(TSN) propose un ensemble de standard connu sous le nom de standard TSN afin d'étendre Ethernet. Ces nouveaux protocoles ont très vite gagné l'intérêt du monde industriel embarqué par la promesse d'un réseau unifié adaptable à de nombreux débits avec de nombreux mécanismes de qualité de service destiné à offrir différent niveau de déterminisme. Parmi les nouveaux mécanismes proposés, quelques-uns, comme le Time-Aware Shaper ou bien le Cyclic Queuing and Forwarding, se démarquent par un besoin de base de temps commune dans l'ensemble du réseau pour fonctionner optimalement. Cette synchronisation est apportée par le standard IEEE 802.1AS qui est l'adaptation pour les réseaux TSN d'IEEE 1588, plus connus sous le nom Precision Time Protocol (PTP).

Cependant, ce type de protocole de synchronisation est une nouveauté conséquente qui doit apporter de nombreuse garanti avant une utilisation dans un système embarqué critique. C'est dans ce contexte critique que se déroule cette thèse avec des besoins provenant du monde automobile, aéronautique et spatial. Ainsi, dans ce manuscrit, nous tenterons de fournir les clés pour obtenir ces garanties.

Dans un premier temps, nous nous poserons la question de la précision de synchronisation atteignable avec IEEE 802.1AS. Pour ce faire, un ensemble d'améliorations apporté à un simulateur open source du protocole, effectué dans le but de le rendre représentatif de la réalité, seront présentés. Puis fort du savoir acquis avec le simulateur, un modèle formel permettant le calcul d'une borne sur la précision pire cas sera explicité. Ces deux contributions seront étayées à l'aide de mesure intensive obtenu sur du matériel supportant IEEE 802.1AS et permettront d'étudier les paramètres dimensionnant de la précision de synchronisation.

Dans un second temps, nous nous interrogerons sur la capacité du protocole à résister aux pannes de lien et d'équipement. Ainsi, une comparaison entre les deux mécanismes proposés par le standard sera effectuée. Cette étude mettra en avant que le mécanisme de robustesse statique répond mieux au besoin des environnements critiques au prix d'une configuration manuel plus complexe. Pour une adoption de ce mécanisme dans l'environnement industriel, il nous a alors semblé important de proposer une méthode pour aider à la configuration de ce mécanisme. Ainsi, une méthode d'évaluation de la robustesse destinée à déterminer la configuration optimale sera proposé.

Et enfin, nous mettrons en pratique, sur le cas d'étude d'un réseau embarqué de satellite, les résultats obtenus précédemment dans le but d'étudier l'impact de la précision et de la robustesse sur le reste des activités du réseau. La faible influence en termes de bande passante perdue à cause d'un surdimensionnement des activités dicté par le temps pour prendre en compte la précision et en termes de latence pire cas des flux applicatifs sera mise en avant. Ces résultats nous permettrons aussi de mettre en avant la nécessité de borner finement la précision pire cas, mais aussi de relâcher le besoin de précision pour la plupart des utilisations du Time-Aware Shaper.

Acknowledgements

Dans ce manuscrit qui conclut cette aventure, je tiens à remercier les personnes qui m'ont soutenues et encouragées durant ces trois années.

Premièrement et avant toute chose, je voudrais remercier Thuy-nhi pour son soutien et sa très grande patience. Je remercie aussi mes parents ainsi que mon frère pour m'avoir toujours encouragé dans mes études.

En second lieu, je tiens à remercier Philippe, Katia et Jean-luc pour m'avoir fait confiance, suivi et guidé dans mes travaux de recherche, mais aussi pour l'ensemble des connaissances et conseils apportés. Ça a été un plaisir de travailler avec vous.

Un grand merci aussi à mes collègues de l'IRT : André, Charlie, Christophe F., Christophe F., Damien, Guillaume B., Guillaume P., Massimo, Morgane, Philippe C. et Philippe P.. Cela fut un plaisir de découvrir et d'explorer TSN avec vous. Ainsi qu'un tout aussi grand merci à mes collègues de l'IRIT, Antonin, Benoit, Edouard, Fabien et Julien pour avoir égayé mes pauses.

Je remercie également l'IRT Saint Exupéry et l'ensemble des membres du projet EDEN qui ont permis à cette thèse de voir le jour et de m'avoir permis de la réaliser. Merci pour le défi que vous m'avez proposé, j'espère que mes travaux vous aideront à avancer. Je vous souhaite plein de réussite dans la suite de vos travaux avec TSN.

Et enfin, mes remerciements vont aux membres du jury pour l'intérêt porté à mes travaux de recherche. En commençant par Nicolas Navet et Ye-Qiong Song pour leur investissement en tant que rapporteur, puis à Marc Boyer, Raphaël Naves et Thomas Watteyne pour avoir accepté de prendre part au jury.

Contents

Part I

Introduction

Today, the buses used to interconnect critical embedded systems are reaching their limits in terms of bandwidth, message size and number of nodes. Some of these buses have already been replaced by networks such as AFDX in aircraft or TTEthernet in certain space vehicles. But these networks were designed for a specific need at the cost of a lack of adaptability to other needs. However, in recent years a set of network standards has emerged with mechanisms that could answer this issue and meet the needs of the critical embedded world. These standards are proposed by the IEEE 802.1 Time-Sensitive Networking (TSN) working group and are commonly referred to as TSN standards. Networks following these new standards promise an Ethernet technology with service differentiation: it can carry a mixed-critical set of flows, with some of them requiring deterministic real-time guaranties, while taking advantage of multiple physical layers and the Ethernet toolbox ecosystem. Some of the new mechanisms standardised by the working group, such as the Time-Aware Shaper (TAS) that controls access to the medium in time, stand out by the need for a common time base in the network. One way of satisfying this need for a common time base in the network is the deployment of the synchronization protocol described in IEEE802.1AS.

Before leveraging such a protocol in a critical embedded environment and carry real-time flows, a number of scientific challenges must be tackled. We have identified two areas where questions remain numerous and guarantees are lacking. The first area revolves around achievable precision. Indeed, although experimental work shows that the protocol can achieve sub-microsecond precision in a 7-hop network, the parameters offering this precision are still poorly understood. The second area concerns robustness to failures. Indeed, the publication of a revision of the standard in 2020 proposes a new mechanism of robustness to failures which seems promising for the world of critical embedded systems, but due to its recent publication, little work has been devoted to it.

Therefore, during the course of this thesis, we have investigated these two areas in order to provide the guarantees needed to consider a safe deployment of IEEE802.1AS in automotive, aeronautical and space embedded networks. In summary, the overall objective of this thesis is to derive mechanisms and models to guarantee the precision and robustness of a synchronization service rolled out in a critical embedded TSN networks.

To meet this objective, we study three scientific questions, each one of them being discussed in one part of this manuscript. The first question we address is whether it is possible to derive a safe bound on the worst-case synchronization precision of IEEE802.1AS in an embedded network. This question is discussed in part II. The second question, discussed in part III, is whether it is possible to provide a robust deployment of the synchronization service in the event of node or link failures. The last question, detailed in part IV, asks if the synchronization service has a noticeable impact on the other flows of the network in terms of bandwidth or timeliness. Before giving our answers to these three questions, we define in part I the embedded context of this thesis and give a digest of the state-of-the-art literature on network synchronization. And a detailed review on precision evaluation and robustness designs of synchronization protocols ends part I.

To derive a safe bound on the worst-case synchronization precision in part II, which represents the main theoretical contribution of this thesis, we start by studying the IEEE802.1AS protocol behavior using simulations and measurements. To ensure that simulations are representative of reality, we propose several improvements to an open source protocol simulation library stemming from an intensive series of experimental measurements. We derive a procedure to calibrate this simulator to any existing Ethernet physical layer technology, and give numerical results for 100Base-T and 1000Base-T. Taking advantage of the various sources of error identified in our simulation and measurement study, we propose a theoretical bound on the worst-case precision achievable with the protocol. Therefore, we extend a formal state-of-the-art model of the worst-case precision bound to encompass all known sources of error, offering a reduced pessimism. As for the simulation study, this bound is challenged by experimental measurements. We show that this new bound model provides

interesting keys to understand and quantify the influence of many parameters on the worst-case precision. The work in this part has led to two publications [39] and [41].

In the third part, we question the robustness to link or node failures of the protocol. This question is central to our industrial partners. In this part, we start by comparing the new robustness mechanism (e.g. external port configuration with multiple domains and hot standby Grandmaster) with the old one (e.g. BMCA), to determine which one is best suited to critical embedded networks. This study involves experimental, formal and qualitative comparisons that lead us to the conclusion that the new mechanism is better suited for embedded networks. Then, we propose to resolve one of the limitations of the new mechanism. Indeed, this mechanism gives the network designer control over the configuration of the synchronization distribution spanning trees. However, there are tens to thousands of possible configurations for our case studies. We therefore propose a method for finding the most precise configuration among the most robust ones. The works presented in this part were presented in two international conferences: [40] and [42].

Finally, in the fourth part, we question the impact of the synchronization service on the network performance. We carry out the analysis on a use case representative of a satellite embedded network. We first investigate the relation between the target level of precision and the bandwidth it costs in terms of time guard bands consumption when using the Time-Aware Shaper (TAS). The analysis extends to the deployment of multiple synchronization domains that are required for robustness. Secondly, we investigate the influence of precision and number of domains on the worst-case traversal time of the rest of flows. With these two studies, we highlight the very low impact of the synchronization protocol on other network activities in this use case.

Chapter 1

Embedded Context

1.1 Embedded networks

Let's start by defining embedded networks, critical embedded networks, then TSN networks and the case studies that will serve as illustrations in this manuscript.

1.1.1 Definition

An embedded network provides the communication infrastructure for distributed embedded systems. "Embedded system" is a term that first appeared in the 1960's during the race to the moon. One of these early systems was the guidance system for the Apollo missions. The following definition can be found on Wikipedia : An embedded system is a computer system—a combination of a computer processor, computer memory, and input/output peripheral devices—that has a dedicated function within a larger mechanical or electronic system. It is embedded as part of a complete device often including electrical or electronic hardware and mechanical parts. Because an embedded system typically controls physical operations of the machine that it is embedded within, it often has real-time computing constraints. These systems are generally constrained by their memory space, computing power, execution time and sometimes autonomy.

So today, the various systems in a telephone or washing machine can be defined as embedded systems. In our case, we're concentrating on so-called critical embedded systems, and more specifically systems in the automotive, aeronautics and space sectors. The notion of criticality comes from the impact of uncontrolled failure, which can lead to the loss of many lives.

1.1.2 A brief historical overview

Historically, these embedded systems were monolithic. Then, they evolved into distributed systems with sensors/actuators located away from the calculator. This has led to the use of dedicated links between different elements of the same system. On such dedicated link computing a bound of traversal delay was trivial. But limitations of scaling such links were quickly reached as the number of systems increased. Dedicated deterministic buses were then proposed to meet different needs. These include CAN[10], Flexray [24], LIN[28] and MOST[35] buses in cars, CAN and MIL-STD-1553[1] buses in aircraft, and MIL-STD-1553 buses in satellites. These buses allow several Electronic Control Units (ECU) to share the same communication medium, reducing the number of links between subsystems. But the needs of new, increasingly bandwidth-hungry, applications

cannot be met by buses reaching a few Mb/s at best. So deterministic and higher-speed networks such as AFDX[15], TTEthernet [56] and SpaceWire[17] have seen the light of day in the aerospace industry. SpaceWire was designed from scratch, whereas AFDX and TTEthernet have chosen to start from classic Ethernet and add mechanisms to guarantee real-time behavior. These technologies are relatively expensive, due to the low demand for this type of product (only a few hundred aircraft per year, and even fewer satellite) and their highly proprietary nature. Against this backdrop, Time-Sensitive Networking (TSN) has emerged. TSN is also based on Ethernet but stands out for its economy of scale, which could come from the automotive sector, and is standardized by the IEEE, making it more open than its equivalents. Section 1.2 describes TSN in greater detail.

1.1.3 Validation and certification

These critical buses and networks differ from conventional networks in their validation and certification process. Depending on the criticality of the information passing over the network, each device must undergo a different validation or certification process. Let's take the example of aeronautics. This process is described by DO-254[11] for hardware development and DO-178[23] for software development, and its aim is to keep the occurrence of failures below a time threshold according to their severity. The classification of failures according to severity and the thresholds are described in Table1.1. To achieve this objective, standards impose rules to be followed according to criticality level. For example, in the case of software development, the certification of a DAL-C system and a DAL-B system is differentiated by the separation of development and validation activities into two independent teams. A DAL-B system and a DAL-A system are distinguished by the need for complete coverage of code conditions, thus prohibiting the existence of dead code. DAL-E is the least critical and DAL-A the most critical level. For the most critical systems, norms require that all requirements described in the system specification be verified. Thus, for a system with the requirement "the system must receive its data from the network periodically, without loss and with a bounded delay", it is necessary to provide proof of the absence of loss and bound latency in the network. To prove these properties on an AFDX (or Ethernet) network, it is possible to perform a formal analysis known as network calculus[69] to determine worst-case bound and jitter on the network transit time but also the buffer occupation in network devices. A network device must be certified for the highest DAL level of the messages passing through it. So if a DAL-C system coexists on a network with a DAL-A system, the network devices will be considered DAL-A.

Automotive systems validation is similar, though less constrained. Systems are classified as ASIL from A to D. ASIL D being the most constrained and equivalent in terms of constraint to DAL-B. Network validation have to be done on CAN and Flexray bus traversal delay since they are used on the most critical car system. CAN and Flexray are by construction deterministic. High priority CAN traversal time bound are trivial to compute whereas lower priority ones can be tricky. However, lower priority CAN messages are less critical and often don't require bound validation. Works [50] have been done to compute worst-case traversal delay even for lower-priority message.

For spacecraft, the level of criticality depends on the mission. A manned mission or the return of a sample from Mars to Earth is more critical than an unmanned mission such as an Earth observation mission. In the remainder of this manuscript, one of our case studies will be an Earth observation satellite. For the network of this type of satellite, only validation is required. Validation is mainly performed via automated tests, but can also be carried out using formal analysis. These analyses are mainly Worst-Case Execution Time (WCET) analyses on the system side. The network is not affected by this timing analysis, since MIL-STD-1553 relies on a simple and deterministic command/response scheme.

DAL	Failure Classification	Failure threshold (in flight hour)	Failure Description
A	Catastrophic	10^9	Failure conditions that would prevent continued safe flight and landing.
B	Hazardous	10^7	Failure conditions that would reduce the capability of the aircraft or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be: a large reduction in safety margins or functional capabilities, physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants.
C	Major	10^5	Failure conditions that would reduce the capability of the aircraft or the ability of the flight crew to cope with adverse operating conditions to the extent that there would be: a significant reduction in safety margins or functional capabilities, a significant increase in flight crew workload in conditions impairing flight crew efficiency, or discomfort to occupants, possibly including injuries.
D	Minor	10^3	Failure conditions that would not significantly reduce aircraft safety, and which would involve flight crew actions that are well within their capabilities. Minor failure conditions may include: a slight reduction in safety margins or functional capabilities, a slight increase in flight crew workload, such as routine flight plan changes, or some inconvenience to occupants.
E	No effect		Failure conditions that do not affect the operational capability of the aircraft or increase flight crew workload.

Table 1.1: DO-178 and DO-214 DAL level description

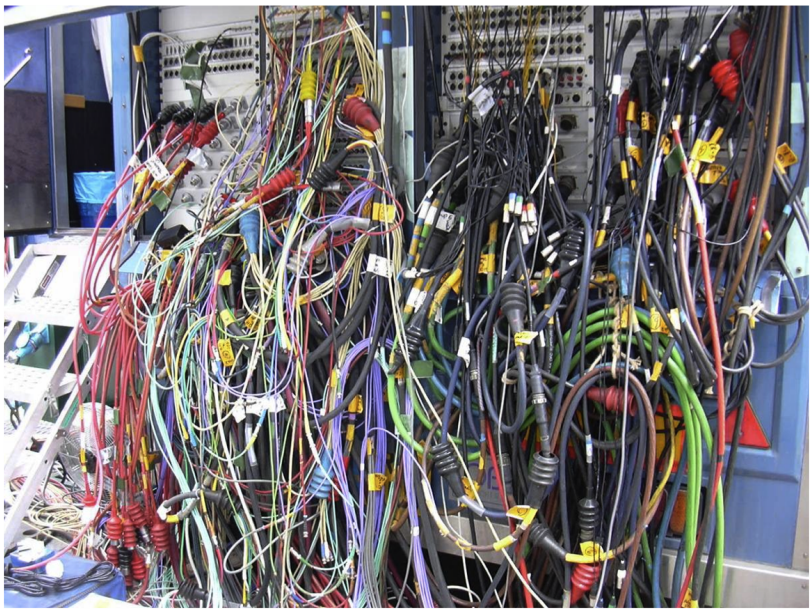


Figure 1.1: Wiring of a patch bay outside broadcasting van. (Source : `https://upload.wikimedia.org/wikipedia/commons/d/df/StudioWiring.png`)

1.2 Time-sensitive Networking

1.2.1 From AVB to TSN

In 2005, the IEEE802.1 Audio Video Bridging (AVB) working group was created with the objective to respond to a need for simplification in the professional Audio/Video (proAV) industry (stage lighting and sound, conference system, multimedia diffusion system or live TV production). Indeed, this industry mainly uses unidirectional point-to-point communication technology such as serial digital interface (SDI) for video that lead to complex wiring patch as illustrated in Figure 1.1. For this purpose of simplification, the working group chose to extend classic Ethernet by adding new QoS mechanisms to guarantee a certain level of determinism and synchronization for applications.

This working group thus proposed a group of six standards commonly called AVB standards in which we find standards such as IEEE802.1Qav-2009 [19] which specifies the use of the well-known Credit-Based Shaper (CBS), IEEE1722-2011 [21] which specifies the Audio Video Transport Protocol (AVTP) intended to synchronize the playback of audio/video content on different receivers (speaker or display) and a synchronisation protocol IEEE802.1AS[22] to allow the use of AVTP.

Quickly, these standards are gaining interest in other industrial sectors because they allow to have a deterministic high bandwidth Ethernet network. And in 2012 the AVB working group became the Time-Sensitive Networking (TSN) working group integrating people from the proAV world but also from industrial automation and automotive world.

1.2.2 Definition, objectives and usages

The Time-sensitive Networking working group proposed a set of standards known as TSN standards. The aim of these standards is to make Ethernet deterministic and thus enable the trans-

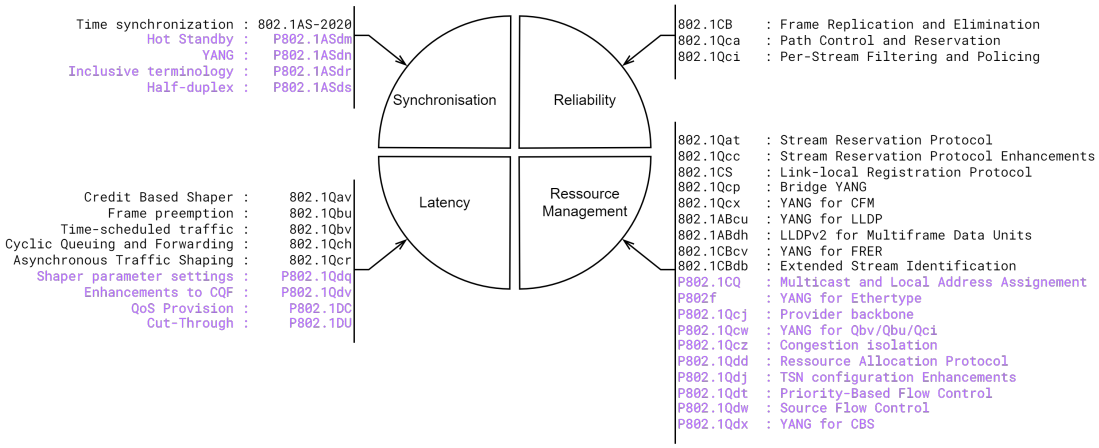


Figure 1.2: TSN standards per group. Ongoing standard are display in purple.

mission of time-sensitive data. These standards can be divided into four groups according to their purpose: Synchronization, Availability/Reliability, Latency and Resource Management. Among the standards dealing with synchronization, we find IEEE802.1AS[34] in its various versions and amendments, which will be studied in this manuscript. In the Availability/Reliability group, we find standards such as IEEE802.1CB[30], which proposes a mechanism to enable message distribution using redundant paths, or IEEE802.1Qci[29], which checks whether a flow complies with its transmission contract when entering a network device. For latency control, there are the Credit Base Shaper (IEEE802.1Qav[19]), Time-Aware Shaper (IEEE802.1Qbv[27]), Asynchronous Traffic Shaper (IEEE802.1Qcr[33]) and Frame Preemption (IEEE802.1Qbu[26]) standards. These different mechanisms make it possible to control access to the medium according to different criteria in order to meet the need for determinism. And finally, the Resource Management group includes standards dealing with the YANG data format, enabling configuration of all the new mechanisms, as well as the Network Resource Reservation mechanism (IEEE802.1Qat[20]). The whole range of TSN standards is summarized in Figure 1.2. To address the needs of each sector, a subset of standards is selected and configurations are recommended in profiles such as P802.1DF for Servive Provider networks, P802.1DG for automotive and P802.1DP for aerospace.

TSN differs from the above-mentioned technologies (except AFDX and TTEthernet) in its dependence on Ethernet. By making Ethernet deterministic, TSN makes it possible to use many physical layers, and especially physical layers with much higher bandwidth than those used today in critical embedded networks. Thus this makes TSN suitable for low-speed devices with 10Mb/s physical layers, but also for the most demanding requirements, up to several gigabits per second. With previous technologies, two different protocols would have been used and interconnected by a gateway to offer different bandwidth and level of determinism. Ethernet components are also much more affordable, thanks to their widespread use in all kinds of sectors. Thanks to Ethernet's 8 priority levels and the possibility of using different medium access mechanisms, TSN is particularly well-suited for networks in which information of varying criticality and need is in transit thereby promising to reduce the number of networks needed in a single system.

In this manuscript, we will focus on IEEE802.1AS and its amendments. However, in order to study the interaction between synchronization frames and the frames of other flows in Part IV, we

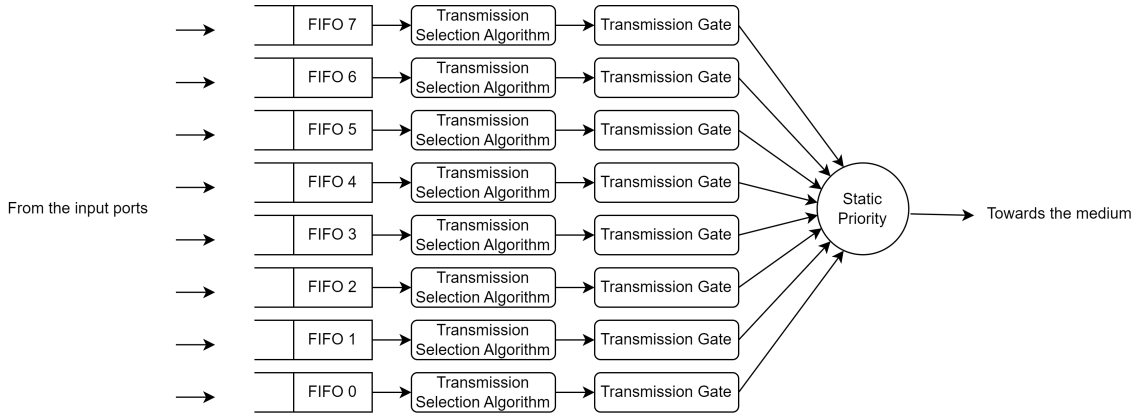


Figure 1.3: Representation of a TSN 8-FIFO output port

need to start by describing the TSN medium access policy, which is an extension of the Ethernet policy. Indeed, a significant part of the delay experienced by a frame crossing an Ethernet network, and by extension TSN, is waiting time in the various output ports on its path. This waiting time depends on the access policy to the medium. Indeed, a frame entering a switch through an input port is directed to an output port by the switching matrix, which places this frame in one of the 8 FIFOs of the output port according to its priority level, as specified in the Ethernet header. Without a TSN mechanism, the frame can only be transmitted if no frame is currently being transmitted, and if the higher-priority FIFOs are empty. This is known as the Static Priority mechanism. With TSN, the output port architecture is shown in Figure 1.3. In addition to FIFO and static priority block, there are two new blocks. The first is the transmission selection algorithm. TSN offers two transmission algorithms: Credit-Based Shaper (CBS) and Ansynchronous Traffic Shaper (ATS). It is also possible to select no transmission algorithm at all. The second is the transmission gate, which is linked to the use of the Time-Aware Shaper(TAS). The TAS controls in time the opening and closing of these gates. If the TAS is not used, all gates are always open. With such a configuration, for a frame to be sent, the transmission selection algorithm, assigned to the frame's FIFO, must authorize sending (e.g. positive credit in CBS), its gate must be open, it must be the highest-priority frame among the candidates for transmission, and the medium must be free. Numerous research works, such as [77], address the choice/combination of transmission selection algorithm and schedule TAS to meet different requirements.

1.2.3 Case studies

The above-mentioned advantages are prompting industrial companies in various sectors to take an interest in TSN. This thesis is part of a project called EDEN, in which Airbus Defence & Space, Airbus Commercial Aircraft, CNES, Continental, Safran Electronics & Defence, Thales Alenia Space and Thales are studying the use of TSN in their embedded networks. The companies can be grouped into three sectors: automotive, aeronautics and space. Each group has proposed a case study representative of the envisaged use of TSN in its sector. These case studies are presented in the remainder of this sub-section and will be used regularly in the manuscript to illustrate our results.

Automotive

The automotive case study, pushed by Continental, is an envisioned topology for the use of TSN in autonomous cars, and aims to respond to the increasing need of bandwidth and the change in architecture. Today's automotive industry is moving towards a zonal architecture. The topology in question is shown in Figure 1.4. In this topology, there are seven zones, each centered around a switch, itself connected to the other switches by one or two links, depending on the zone's level of criticality. The topology is structured around two central zones: the Performance Computing Unit (PCU) and the Performance Computing Unit Autonomous Driving (PCU AD), where the majority of flows converge. End stations include environmental reconstruction devices such as cameras, radars and lidars, as well as various displays and host computers. The core network and greedy end stations are at 1Gb/s and the rest at 100Mb/s. On this topology, the following four flow groups transit:

- CAN over Ethernet : CAN messages translation over Ethernet for legacy systems. These flows are very small, periodic and have top priority
- A/V : Audio and Video flows are used for environment reconstruction and for infotainment. These flows are bursty, periodic with high to low priority
- Some/IP : Scalable service-Oriented MiddlewarE over IP is an automotive middleware solution that can be used to exchange messages with dynamic subscription to topics. These flows are small, periodic and top priority.
- Best effort : Messages used for Firmware/Software update and diagnostic tool. These flows are large and aperiodic but have low priority.

In this case study, synchronization allows the use of Time-Aware Shaper on the most constrained flows, as well as the use of critical distributed applications such as light control and less critical ones such as infotainment through the AUTOSAR and AUTOSAR ADAPTIVE API.

Aeronautic

For the aeronautical case study, EDEN project members chose to implement TSN on a less critical system than the current AFDX network. It concerns the digitalization of cockpit audio and was proposed by Airbus Operation, Thales Avionics and Safran Electronics & Defence. The topology studied is shown in Figure 1.5. Six switches interconnect nine end stations in this 1Gb/s networks. These end stations are detailed in the following list:

- D-ACP CAPT : Captain headphone with microphone that sends and receives periocally audio through the network interface. This device also transmits control data when Push To Talk (PTT) button is pressed.
- D-ACP F/O : Same as D-ACP CAPT for the First Officer (copilot)
- SDR BCU : In charge of VHF control and communication channel management.
- VHF 1 and 2 : Redundant VHF Radio for communication with Air Trafic Control (ATC)
- SATCOM : Communication means able to deliver/transmit audio and data between ground station and aircraft, using satellite constellation when VHF communication is not possible.
- ACM : Audio Communication Means is a touch screen panel that allows pilots to select frequencies to be operated by VHF 1 and 2.
- GPS receiver 1 and 2 : Act as redundant reference clock for audio synchronization playback.

On the traffic side, most of the bandwidth is used to transfer audio between the D-ACPs and the receivers. The rest of the bandwidth is used for control traffic sent by the PTTs of the D-ACPs, ACM and SDR BCU.

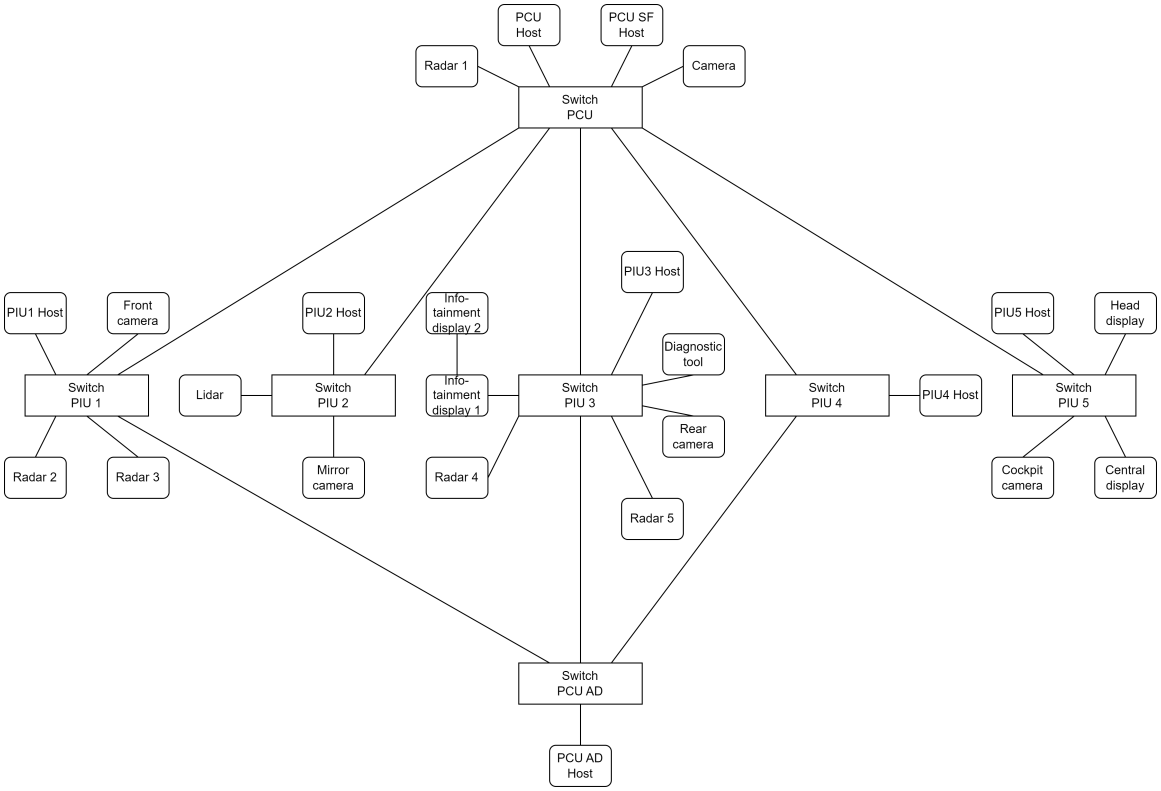


Figure 1.4: Illustration of the TSN automotive case study topology

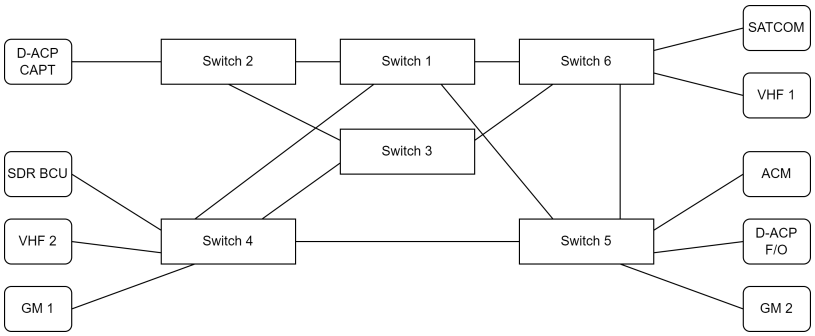


Figure 1.5: Illustration of the TSN digital audio case study cockpit topology

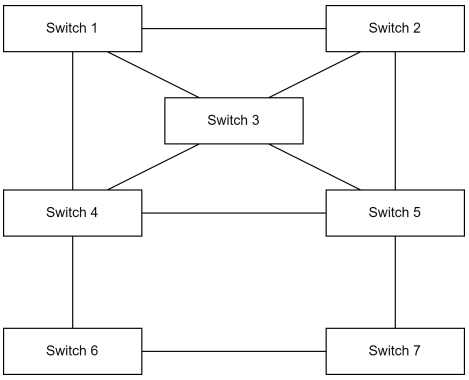


Figure 1.6: Illustration of the A350 AFDX case study topology

This case study aims to investigate the use of the AVTP protocol, which relies on synchronization to enable synchronized playback of multimedia content (in this case, audio) on multiple receivers (in this case, D-ACPs). To ensure that desynchronization of such multimedia content playback is not detected by a human, synchronization must achieve a precision of less than 5ms.

In addition to the digital audio case studied as part of the EDEN project, the topology of the A350's AFDX network will be regularly used to illustrate our results. Indeed, this topology stands out from the other case studies in terms of its size and connectivity. It also allows us to consider the use of TSNs on the aircraft's most critical network. This topology is illustrated in Figure 1.6. We'll concentrate on the core network, as the end stations don't add anything to the connectivity of the topology. As the flows traversing this topology are not public, it will only be used to study how our algorithms scale.

Satellite

This case study is a conversion of the current earth observation satellite architecture, i.e. MIL-STD-1553 for the platform and SpaceWire for the payload, into a single TSN network architecture. It was proposed by Airbus Defence & Space, CNES and Thales Alenia Space. The topology is shown in Figure 1.7. Here, each device is duplicated (cold redundancy) due to the difficulty of repairing satellites in flight. Paths are also duplicated in the same way as done for the MIL-STD-1553 bus on the current platform. The architecture is centralized around the On Board Computer (OBC), which will trigger the acquisitions and control the actuators. As with the previous case studies, this case study is characterized by variable-priority flows ranging from best-effort to real-time. This case study differs from the others in that its latency and jitter constraints are much lower than those of the others, since it's the only case study where the control/command messages use the TSN network. The hardest constraint is jitter on the network traversal time of 1µs. The only latency constraint in this domain is that the message must arrive before the end of the cycle in which it was sent. The latency constraint then depends on the instant of transmission of the flow and therefore varies according to the emission schedule chosen.

To meet this 1µs requirement, the Time-Aware Shaper is a very promising candidate. To do this, it is necessary to achieve sub-microsecond synchronization precision in this network. Because of this particularity, this is the most constrained case study in terms of synchronization precision that we will study in this manuscript.

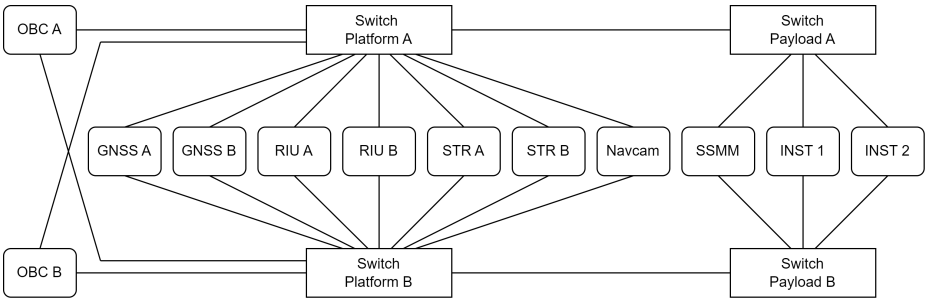


Figure 1.7: Illustration of the TSN satellite case study topology

Chapter 2

Synchronization

2.1 Synchronization: definition, objectives and usages

The act of synchronizing, or synchronization, can be defined in many ways due to its variety of uses. The dictionary "Le Robert" defines the verb "to synchronize" as to make actions happen simultaneously. However, such simultaneous actions can only occur if the actions have a common cause, by chance or by a common knowledge of time. This last point is the one that interests us in this manuscript. This concept is found in one of the Oxford dictionary definitions which is the following: adjustment of a clock or watch to show the same time as another. This notion of time is also found in the ethymological origin of the word in Greek. Indeed, synchronization is the combination of the word sun (together) and khrónos (time).

The need for synchronization stems from the fact that all clocks (biological, mechanical or electronic) have their own perception of time. So two clocks that share the same notion of time at one moment will end up not sharing it at all. We say they drift. To compensate for this drift, clocks can be synchronized on a regular basis.

In practice, synchronization is present everywhere in nature from the synchronization of neurons to the singing of crickets. For humans too, from the simultaneous attack of prehistoric animals to the synchronization of audio and video in modern cinema, as well as in dance and music or even in the use of tools requiring more than one person, such as the two-man saw.

However, to meet modern needs, synchronization had to become significantly more precise. Indeed, let's start by going back 80 years with a photography 2.1 of the Tonga operation (landing of British airborne troops in the night of June 5 to June 6, 1944 in Normandy). In this photo, four paratroopers are seen synchronizing their watches to share a common notion of time. This manual synchronization of the watches allows at best to reach a precision of several tens of seconds.

Then, with the digitization of communications, even more constrained precision needs appeared. We can take the example of the Musical Instrument Digital Interface (MIDI) bus which appeared in the 1980's, allowing communication between multiple electronic instruments and/or music software. On this bus, a master distributes the tempo to all the slaves with 24 "MIDI clock" messages between two black notes.

Synchronization was also essential for the advances in localization like GNSS (GPS, GLONASS, Galileo and Beidou). Indeed, the more and more precise synchronization allowed to measure the distance covered by signals going at the speed of light more and more precisely.

Today, synchronization is even more integrated in our everyday lives, from stock exchange to digital telephony, including train and electric grid control, data encryption using certificates and

Figure 2.1: The Pathfinders of the 6th Division synchronizing their watches on June 5th before being parachuted into Normandy. (Source:`https://upload.wikimedia.org/wikipedia/commons/4/45/Operation_Tonga.jpg`)

time setting of computers, telephones and other smart devices.

2.2 Synchronization in and for network

As synchronization is used for a wide range of applications, let's focus on networks. For the latter, synchronization has many uses, which can be grouped into two categories: for networks and for applications.

The main use of synchronization for networks is Time Division Multiple Access (TDMA). TDMA is a medium access technique that allows multiple streams to share bandwidth in time. Each stream is allocated one or more transmission opportunities, called slots or windows, in a repeating cycle. This access method is regularly used in wireless networks (such as GSM) or buses (such as FlexRay or 10Base-T1S) to avoid collisions on shared medium. Despite the absence of collisions, this access method is also used in full-duplex networks such as TTEthernet or TSN to act as a shaper that determines which FIFO an Ethernet output port can transmit. This ensures communication windows for certain flows, reducing their latency and/or jitter. Another network application emerging with the IEEE802.1Qci TSN standard is time policing. Indeed, this standard proposes a time-based flow monitoring mechanism to check that messages in the flow have arrived within a certain time window. Various actions, up to and including dropping the packet, can be taken by this policer. And last but not least, synchronization can be used to enable a network-wide schedule of transmissions to perform temporal load balancing that can reduce network latency.

As far as applications are concerned, there are several types of use. The first concerns distributed applications. When an application is distributed over several network nodes, it is sometimes necessary to date an acquisition or an action. Such dating can, for example, be used to correlate data from different sensors, or to coordinate different actions in phase or with a contrasting phase shift. An example of the latter from the automotive case study is the control of a car's headlights. To switch on the front lights, the command is sent by the central ECU to two controllers (one on each side). The specification of this function requires the two lights to switch on with less than 1ms difference. As the time taken to send the command message varies according to the path, it is necessary to indicate in the message the time at which the command will be executed. Another common use of synchronization is for localization purposes. GNSS, the reference for localization, produces imprecise results in places with poor sky visibility, such as indoors, or in densely populated urban areas. This has led to the emergence of new, more localized and more precise solutions, for example for locating robotic forklifts in factories. These solutions, based on wireless communication (WiFi, 5G, UWB, ...), measure the time of flight between the device to be tracked and several devices whose position is known, then by triangulation determine the position of the device to be tracked. For such a time-of-flight measurement, the source and destination must share a common time base.

The various protocols that exist for synchronizing network devices operate in a similar way, when their aim is to achieve a precision lower than the network traversal delay. It relies on two mechanisms. The first of these aims to estimate the traversal delay. The second mechanism distributes a clock top to which the propagation delay estimated using the first mechanism is added, in order to correct the clock. If the required precision is greater than the network traversal delay, only the second mechanism is needed, making the protocol simpler.

2.3 Vocabulary

Next we define some terms that are essential for the rest of this manuscript.

Time and frequency synchronization To begin with, we can define two types of synchronization. The first is frequency synchronization. The aim of this type of synchronization, known as syntonization, is for two things to share the same frequency. This type of synchronization is used, for example, to synchronize RF transmitters and receivers so that the receiving device can read the data at the same frequency as they are sent, thus avoiding the risk of reading the same symbol twice or missing one. This phenomenon is known as bit slip. The second type is time synchronization. This is synchronization in phase and frequency. The aim is for two things to share the same frequency and the same time of the day. An example of this type of synchronization is computer time-keeping.

Accuracy and precision To quantify synchronization, two terms collide: accuracy and precision. These two terms are defined in the International Vocabulary of Metrology[76] and commonly explained using target representations, as shown in Figure 2.2. In engineering terms, the accuracy of a measuring system is the degree to which measurements of a quantity are close to the true value of that quantity. Thus, in the illustration, a system is said to be accurate if its readings are close to the center of the target representing the true value (i.e. obtained with a perfect measurement). Whereas, the precision of a measuring system is the degree to which repeated measurements under unchanged conditions give the same results. In the illustration, a system is said to be precise if these values are grouped together. So, in the case of time synchronization, the center of the target is perfect time and the points are the time of the devices. We don't have the means to measure perfect time with an accuracy of more than a few tens of nanoseconds with the equipment we have at our disposal, whereas we carry out offset measurements between the times of two devices of the order of a few hundred nanoseconds. In view of this ratio, we'll be talking about precision in the rest of this manuscript. For example, if Bob synchronizes his watch with Alice's watch and, after this synchronization, Alice's watch shows 12:00 and Bob's watch shows 12:01, we'll say that the precision of this synchronization is one minute.

Clock The part of the network device that keeps time is called the clock. One of the dimensional parameters of a clock is its drift. All clocks gradually drift away from perfect time. Only periodic synchronization can limit drift. Thus, the time t of clock i deviates from the perfect time t_p due to a drift that varies over time $\rho(t)$ as described in the following equation :

$$\begin{aligned} t_i(t) &= t_p(t) + t_p(t) \times \rho(t) \\ &\text{with} \quad t_i(0) = t_p(0) \end{aligned} \tag{2.1}$$

$\rho(t)$ can either be positive (so clock i can go faster than perfect time) or negative (so clock i can be slower perfect time). The drift evolution over time in the short term is due to changes in temperature, pressure or even supply current, and in the longer term is due to aging. An example of the evolution of t_i compared to t_p is pictured in Figure 2.3. In this example, we see a slow variation of the drift as a function of time, which could be caused by temperature variations around the clock. There are also higher-frequency variations that could be caused by variations in the clock's current supply. Work has been devoted to the study of the different noises that impact different types of clocks, as summarized by Riley et al. in [82].

In the clock datasheet, a bound is generally given on the maximum drift the component can undergo. This bound is expressed in parts per million (ppm). For example, a clock with a bound drift of 10 ppm can drift up to 10 seconds every 1 million seconds, either positively or negatively, from the reference time.

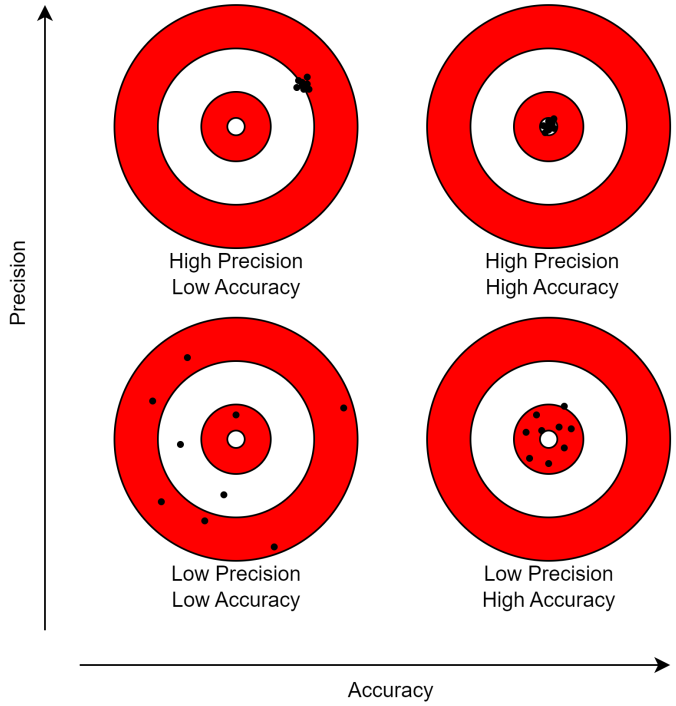


Figure 2.2: Graphical representation of the relationship between accuracy and precision

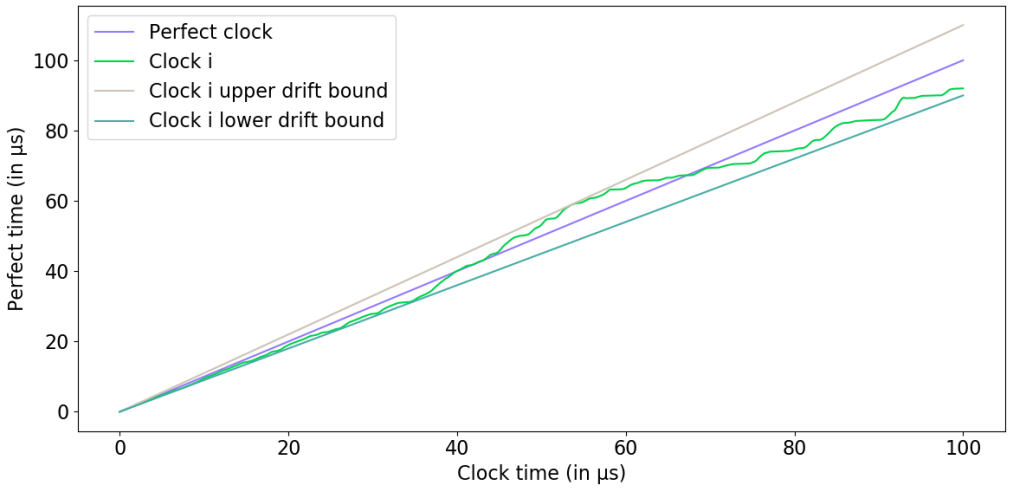


Figure 2.3: Exaggerated example of drift variation as a function of time for clock i

A clock consists of an oscillator that oscillates periodically and a counter that counts the oscillations. The most common types of oscillator are listed below.

- Basic crystal oscillator (XO)
- Temperature Compensated Crystal Oscillator (TCXO)
- Oven Controlled Crystal Oscillator (OCXO)
- Atomic clock

Basic crystal oscillators are electronic components that generate a frequency by electrically energizing piezoelectric crystals (often quartz). These components are very inexpensive (a few cents) and are used to keep track of time, to generate a stable frequency signal used by many digital integrated circuits, and to stabilize the frequency of radio transmitters/receivers. These crystals can be found in countless devices, from digital wristwatches to car ECUs. The bound on drift of such crystals is generally between 10 and 100ppm, i.e. between 0.9s and 9s of drift per day at most.

TXCOs are also crystal oscillators, but are temperature-compensated. In practice, a variable-capacitance diode is used to correct the oscillator frequency as a function of temperature. This improvement makes it possible to achieve drift up to the order of 1ppm, or 90ms per day.

OXCOs are yet another evolution of quartz oscillators, but this time, instead of compensating for temperature, they control it. To do this, the oscillator is placed in a hermetically sealed enclosure with a heating system that heats to a very stable temperature, higher than that possible in the enclosure's external environment. Such an oscillator is much bulkier, heavier (200-500g vs. 20-50g for XO and TCXCO) and more expensive than previous crystals. On the other hand, they can achieve maximum drifts of 1ppb (0.001ppm) or 90µs per day.

Atomic clocks are the benchmark for stability, with a bound on drift of less than 0.001ppb or 90ns per day. They use the perennity and immutability of the frequency of electromagnetic radiation emitted by an electron in an atom as it passes from one energy level to another to ensure the accuracy and stability of the oscillating signal they produce. The elements typically used in atomic clocks are cesium and rubidium. Today, such clocks are the guarantors of time bases such as Coordinated Universal Time (UTC) or International Atomic Time (TAI). However, they can range in size from that of a desktop computer to that of a car, and can be exorbitantly expensive.

In the world of synchronization, GPS clocks are also regularly referred to as GNSS clocks. These are clocks synchronized to the time broadcast by GNSS constellations (GPS, Galileo, Glonass, ...). Indeed, to determine a position from GNSS, it is necessary to be finely synchronized to the same time base as the constellation's satellites. Since GNSS constellation satellites carry atomic clocks, synchronizing to this time base is an interesting alternative to having to embed atomic clocks in our on-board systems. Various device synchronized to this time base. GPS clock drift (i.e. when it loses the GPS synchronization signal) is highly variable. Indeed, basic oscillators (with a drift of up to 100ppm) can be found in low-cost device such as cell phones and car GPS systems. But TCXO and OCXO can also be found in devices designed for datacenter synchronization, or in very precise measuring devices like the Meinberg microSync HR we use in our work.

2.4 Network synchronization protocol evolution

To meet these needs and replace links dedicated to synchronization, numerous network protocols have emerged. The protocols that led to IEEE802.1AS and its competitors are detailed below, in chronological order of appearance. A summary for wired protocols is given in Table 2.1.

Daytime and time protocols

The first synchronization protocols date back to the early days of the Internet. In May 1983, two RFCs were published: RFC867 [2] and RFC868 [3]. RFC867 proposes a protocol called Daytime Protocol. RFC868 describes the Time Protocol. Both are designed to be encapsulated in UDP or TCP frames. However, they differ in the TCP and UDP ports used, and in their time encoding format. Daytime Protocol transmits a human-readable ASCII format such as "Tuesday, February 22, 1982 17:37:43-PST" or "02 FEB 82 07:59:01 PST", while Time Protocol transmits the number of seconds since midnight on January first 1900.

Both protocols operate on the client/server principle. A client needing the time sends a request to a server that knows it. The server replies with a packet containing the time in the protocol format. In terms of precision, these protocols achieve relative precision as a function of the propagation time between client and server. Indeed, if the message takes 10 seconds to reach the client from the server, then once the clock correction has been applied to the client it will be 10 seconds behind the server's time.

NTP

Network Time Protocol (NTP) can be considered as the father of synchronization on the Internet. In 1985, its first version was proposed in RFC958 [5] in the context of ARPA-Internet. It is defined by this RFC as "a protocol for synchronizing a set of network clocks using a set of distributed clients and servers" with the aim of sub-second precision. This difference in precision with its contemporaries is explained by the addition of a mechanism for estimating the time taken by the time message to pass through the network. This estimate is then added to the time written in the packet to take into account of the message's travel time before the clock is corrected. In addition, NTP offers a time distribution architecture.

The proposed architecture is based on a stratum organization as illustrated in Figure 2.4. From top to bottom, we find the devices of stratum 0. These devices have a very precise knowledge of time. These are usually atomic clocks or GPS clocks. Then there are the so-called stratum 1 servers. These servers are directly synchronized to stratum 0 devices via dedicated links such as 1Pulse Per Second (PPS). This time base is then distributed to stratum 2 servers (who act as clients during this exchange) using the message exchange proposed by the RFC. The stratum 2 server then distributes to the stratum 3 server, and so on. Moreover, it's possible for a server in stratum 2 or higher to ask several servers in the previous stratum for their time, in order to check that it's working correctly. This verification can also take place between servers of the same stratum, as illustrated by the horizontal arrows. This architecture can be illustrated in the case of maintaining the time on my work computer. Indeed, it synchronize with the IRT SAINT EXUPERY NTP server, which itself is synchronized with four servers in the fr.pool.ntp.org pool.

In addition to this architecture, NTP differs from its predecessors in that it takes propagation delay into account, as mentioned previously. This mechanism is based on a periodic request/response exchange, as illustrated in Figure 2.5 encapsulated in UDP messages. Although the exchange is identical to Daytime and time protocols, the difference lies in the data carried by these messages. Indeed, the message carries the timestamp for sending the request (t_1 in the figure), the timestamp for receiving the request (t'_1 in the figure) and the timestamp for sending the response (t'_2 in the figure). Using the previous timestamps plus the response reception timestamps (t_2), the client can compute the round-trip delay, noted δ, and the difference between the clocks, noted θ , using the following two equations:

$$\delta = (t_2 - t_1) - (t'_2 - t'_1) \tag{2.2}$$

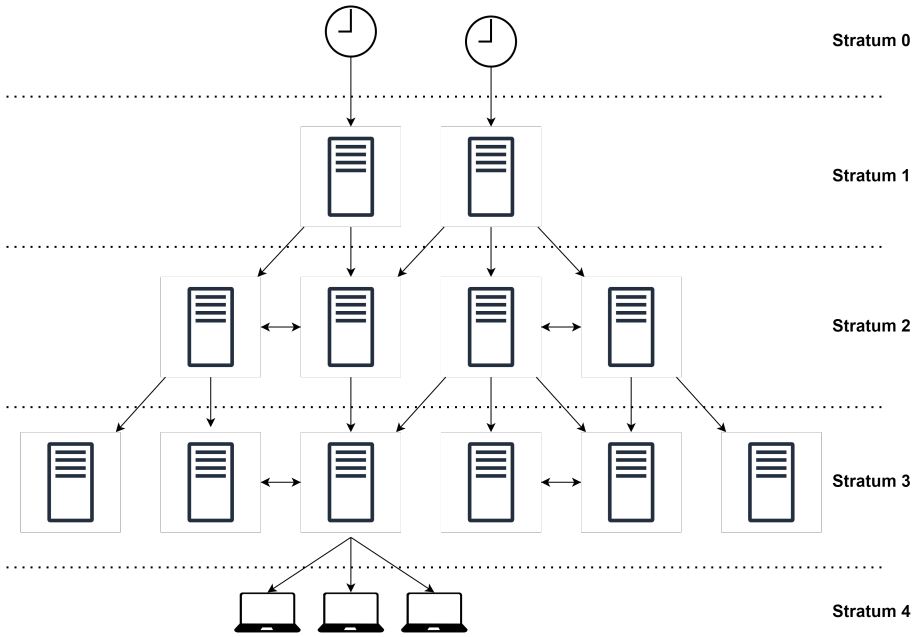


Figure 2.4: NTP stratum architecture

$$\theta = (t'_2 + \frac{\delta}{2}) - t_2 = \frac{t'_1 + t'_2}{2} - \frac{t_1 + t_2}{2} \tag{2.3}$$

RFC958 was subsequently complemented by RFC1059 [6] (NTPv1), RFC1119 [7] (NTPv2), RFC1305 [9] (NTPv3) and RFC5905 [75] (NTPv4). NTPv1, published in 1988, offers a much more complete description of how NTP works, and also proposes various algorithms based on the experimental results described in RFC956 [4]. A year later, in 1989, NTPv2 introduced a management protocol and cryptographic authentication scheme. In 1992, NTPv3 introduced a new procedure for selecting the server to use, based on Marzullo's algorithm. This new version also features an analysis of error sources impacting the accuracy of the reference clock up to the final client. The current NTPv4 version, released in 2010, takes the state machines and pseudo-code of the algorithms described in previous versions and updates them with the new mechanisms, while retaining backward-compatibility. This version also marks the merge of Simple Network Time Protocol (SNTP) with NTP. SNTP is a protocol introduced in 1992 by RFC1361 [78] to meet the synchronization needs of very simple equipment that doesn't have the resources for long-term status storage as required by earlier NTP versions.

All this makes NTP the Internet's synchronization protocol. Its architecture is perfectly suited to this, as it is both centralized around high-precision equipment (stratum 0 and 1) and distributed thanks to the existence of multiple servers in each stratum. Furthermore, since the server requires no computation and no need to keep information on the requester, it's also possible to respond to a very large number of clients with a relatively simple server. However, NTP's precision is severely limited by the assumption of symmetrical propagation times (divide by two in equation (2.3)). This is highly unlikely on networks where a message may take a different route, or be subject to longer buffer times depending on direction, as is the case on IP networks. The research community is still active on this last point, as well as on cybersecurity aspects, as shown by the work of Mkacher et

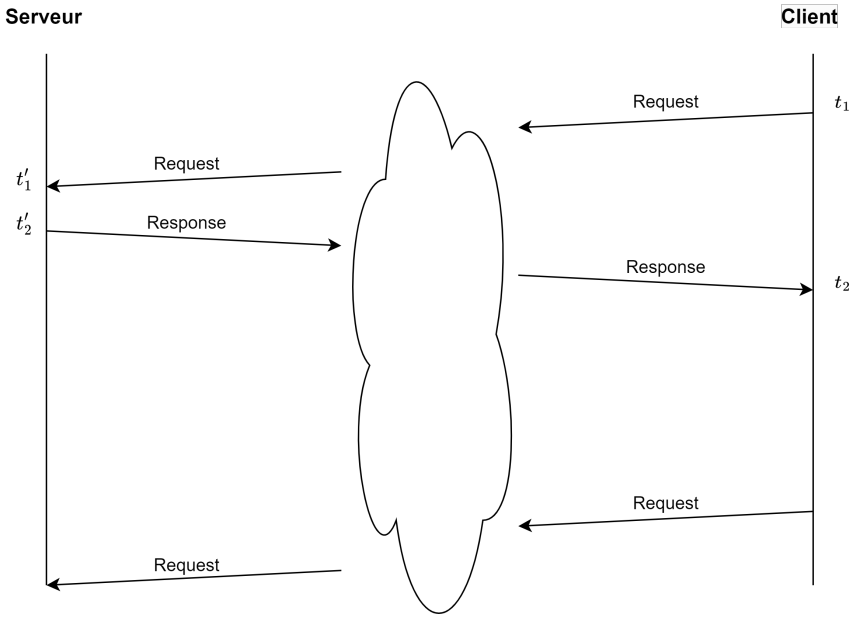


Figure 2.5: NTP message exchange pattern

al. [79].

PTP

In 2002, the first version of the Precision Time Protocol (PTP) was described by IEEE1588-2002 [12]. This new protocol aims to offer a more precise alternative to NTP (for applications that need it) and an alternative to satellite synchronization such as GNSS. For the latter, a fully GNSS-synchronized network would require a GNSS antenna on each of the network devices, which is unfeasible for reasons of access to good-quality signals and/or cost. This first version has been replaced by IEEE1588-2008 [18], known as PTPv2. However, PTPv2 is not backward-compatible with the original version, mainly due to changes in certain data formats such as timestamps. PTPv2 was then enhanced by the current IEEE1588-2019 [32] version, known as PTPv2.1 due to its backward-compatibility with PTPv2.

Unlike NTP, which synchronizes on client request, PTP works on the basis of a periodic distribution emanating from the network's time reference, called the Grandmaster. This Grandmaster is dynamically elected by the Best Master Clock Algorithm (BMCA), a distributed algorithm, by comparing the synchronization quality claimed by the various potential Grandmasters. Slave devices are synchronized using the message exchange described in Figure 2.6 and encapsulated in UDP messages over IPv4 or IPv6. A first message called `Sync` is sent at time t_1, and received at t_2 by the slave. To transmit t_1, the Grandmaster uses a second message called `Follow_Up`. This second message is used to transmit the actual time at which the `Sync` was sent, unlike NTP, which estimates the time at which the message should be sent when it forges the message. The slave responds to the `Sync` with a `Delay_Req` message sent at t_3 and received at t_4. t_4 is transmitted to the slave via a `Delay_Resp` message. This exchange is carried out periodically at intervals in seconds, using the

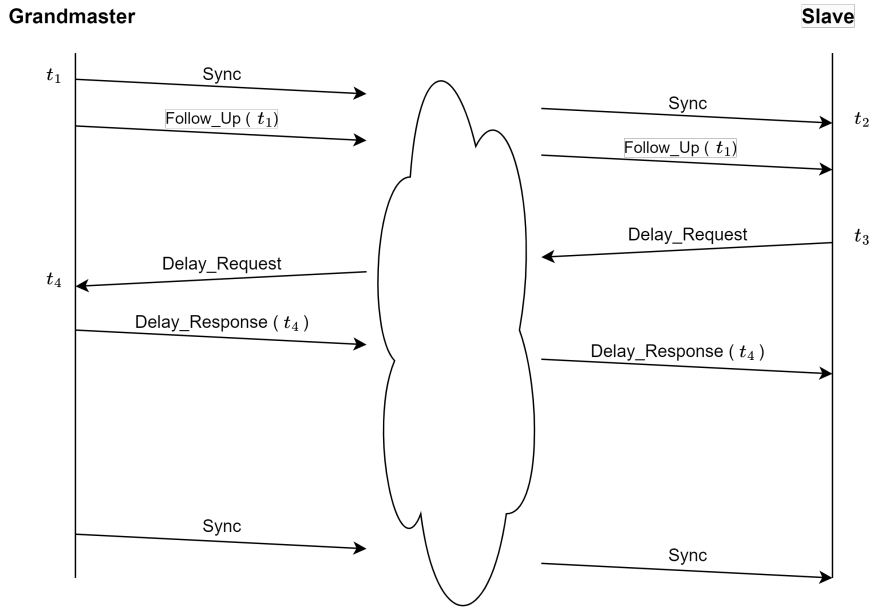


Figure 2.6: PTP message exchange pattern

following values: 1, 2, 8, 16 and 64.

Using these four timestamps, the slave calculates the propagation delay, noted D, between itself and the Grandmaster, as well as the correction offset to be applied to correct its clock, noted θ, using the following two equations:

$$D = \frac{(t_2 - t_1) + (t_4 - t_3)}{2} \tag{2.4}$$

$$\theta = t_2 - (D + t_1) = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} \tag{2.5}$$

One of the new features of IEEE1588-2008 is the introduction of a new delay measurement mechanism: the peer delay mechanism. The major difference between this mechanism and the Delay request-response (or more commonly end-to-end delay mechanism) described above is that it executes peer-to-peer in order to measure port-to-port propagation time. This mechanism will be described in detail in section 2.5. In addition, PTPv2 introduces logical syntonization, which takes into account the difference in frequencies between two devices using the *neighborRateRatio*, also described in detail in the following section. It also features the one-step mode, which consists in sending t_1 in the `Sync`, i.e. without the `Follow_Up` like the two-step mode introduced in PTPv1. The concept of transparent clock is also introduced. This mechanism, which is used for equipment between a Grandmaster and a slave, i.e. switch or router, enables this device to measure the time a message has passed through the device in question. This time can then be added to the message or to the `Follow_Up`, depending on the mode used. In a way, it's a lighter version of the protocol that doesn't allow you to synchronize the device on the route (because you don't need to), while at the same time increasing precision. For example, a propagation time that increases transiently due to a buffer filling on a switch output port can be taken into account when calculating the offset

between the Grandmaster and slave clocks. The possible message interval range was also increased to suit more use cases. Another new feature is the addition of domains, enabling several timebases to cohabit in the same network. The last major new feature is the addition of support for encapsulation in Ethernet, DeviceNet, ControlNet and PROFINET messages.

These new, sometimes incompatible, mechanisms have led to a system of application-specific profiles. IEEE1588-2008 has become a menu from which profiles draw mechanisms and define intervals for the various parameters that meet their needs. Among the best-known profiles are the following:

- Delay Request-Response Default PTP profile
- Peer-to-Peer Default PTP profile
- G.8275.1 made by ITU-T and destinated to the telecoms
- SMPTE ST2059-2:2021 made by SMPTE for the professional broadcast industry
- IEC 62439-3 made by IEC for industrial automation
- IEEE802.1AS made by IEEE TSN working group

The IEEE802.1AS profile will be detailed in the next section.

The latest version of the standard, IEEE1588-2019, takes many of the mechanisms proposed in the profiles and integrates them into the basic standard. These include, for example, the media-dependent and media-independent layer concepts introduced by IEEE802.1AS. The list of proposed mechanisms has also been extended to include Common Mean Link Delay Service, PTP integrated security mechanism, Alternate timescale offset, Mixed multicast/unicast operation, Acceptable master table and external configuration of a PTP Instance's Port state. This standard also brings changes to existing mechanisms, such as the one-step and two-step modes, which switch from device parameters to port parameters, enabling simultaneous support of both modes on a single device. A coefficient has also been added to compensate for asymmetries in propagation delay measurement. The usefulness of this coefficient is limited to asymmetries that can be estimated in advance during a calibration phase, for example.

Today, PTPv1 is rarely used, due to its greater complexity and lower performance than NTP. PTPv2 and v2.1 are widely used in a wide variety of contexts (and where NTP doesn't meet the need), thanks to the profile principle. For example, they are used in Internet Service Provider(ISP) networks to synchronize 5G cells, in datacenters to synchronize certain servers, in laboratories to synchronize measuring instruments and, of course, in TSN networks. Because of these very varied profiles, it's very difficult to assess the precision of PTP, but we can give some bounds. The latter ranges from a few seconds for the least demanding profiles to a few hundred nanoseconds for profiles requiring very high precision.

Synchronous Ethernet

In 2006, ITU-T proposed Synchronous Ethernet, or SyncE, via the recommendations ITU-T Rec. G.8261, ITU-T Rec. G.8262 and ITU-T Rec. G.8264. This protocol aims to syntonize network devices from a reference source. Frequency synchronization is very common in the telecommunications sector, to prevent a phenomenon known as bit slip. Bit slip is the gain or loss of a bit due to clock drift between transmitter and receiver. SyncE is used to synchronize the network core and propagate it to the various wireless telephony antennas.

SyncE uses encoding on the physical line to transmit the frequency, which is then recovered by a Phase Locked Loop (PLL) on the other side of the link, as illustrated in Figure 2.7. This method achieves sub-50ns accuracy, but requires specialized hardware. This precision is achieved without sending a message, but in order to meet the clock source's need for traceability, a message called Ethernet Synchronization Message Channel (ESMC) is periodically broadcast by this source.

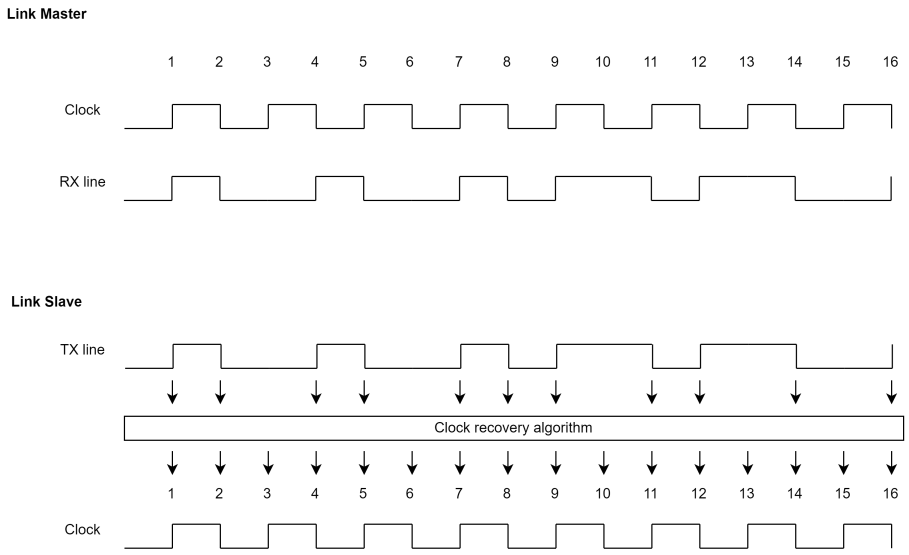


Figure 2.7: SyncE port to port frequency synchronization mechanism

Devices designed to play the role of gateway between Ethernet and the historical medium of the telecom world's core networks, such as SDH, enable frequency synchronization to be passed from one network to the other without difficulty, thanks to their very similar operating modes. This avoids having to replace all the historical network.

White Rabbit

White Rabbit (WR) [84] is a high-precision Ethernet-based synchronization protocol pioneered in 2008 by CERN and the GSI Helmholtz Centre for Heavy Ion Research. Since the release of IEEE1588-2019, it is now one of the profiles described in the PTPv2.1 standard. This profile enables sub-nanosecond precision. It was originally designed for the control and acquisition of synchronized data for CERN's particle accelerators as well as in GSI's Facility for Antiproton and Ion Research. But it is now used in other networks requiring such precision, such as the KM3NeT neutrino telescope or the Large High Altitude Air Shower Observatory. It's an open source, firmware and hardware project.

To achieve such precision, the medium used is fiber optics. This means that, during the calibration phase, the link's asymmetry can be estimated more precisely than with twisted copper pairs, because it's linked to the laser's wavelength and the fiber's optical properties. Precise knowledge of these asymmetries improves the quality of PTP propagation delay measurements. Another improvement is the transmission of time by PTP with syntonization performed by SyncE, making syntonization far more precise than PTP's logical syntonization (at specific hardware cost). With such precise frequency synchronization, the period of PTP exchanges can be greatly increased, offering precise timing with minimal bandwidth use.

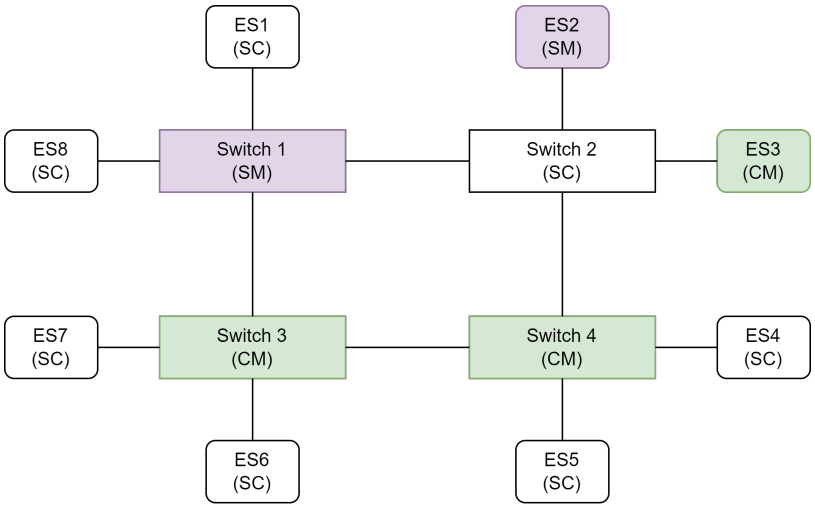


Figure 2.8: TTEthernet device type illustration

TTEthernet

Time-Trigger Ethernet, or TTEthernet, is the protocol proposed by SAE standard AS6802 [56] in 2011. It aims to enable the use of Ethernet in a critical context by providing the necessary guarantees. To achieve this goal, TTEthernet relies on the following three types of traffic:

- Best effort : Lower priority traffic with no delay guarantees
- Rate constrainted : Bandwith guarantees and used for less stringent determinism and real-time traffic requirements.
- Time triggered : Highest priority with predefined (scheduled) emission times for strictly deterministic traffic.

To enable time-triggered traffic, complete network synchronization is required. This synchronization is also defined by SAE AS6802, and is based on a convergence algorithm towards a single time base. To achieve this, there are three types of device in the network, as shown in Figure 2.8. Synchronization-critical devices, such as the Synchronization Master (SM) and Compression Master (CM), are generally redundant to withstand failures. The synchronization process is initiated by the Synchronization Masters, in purple in the figure, distributing the information required for synchronization to the Compression Masters, in green in the figure. The CMs compress the time of the various SMs and correct their clock. The CMs then redistribute the compressed time to the SM and Synchronization Client (SC), in white in the figure, so they can then correct their clocks. The compression step consists of averaging the times, with the possibility of excluding extreme values. This convergence mechanism makes TTEthernet very different from other synchronization protocols, which are based on a hierarchical approach with a central time reference device.

For the distribution of the information required for synchronization, TTEthernet also relies on a very different mechanism from that seen previously with other time synchronization protocols. This is illustrated in the Figure 2.9 in the case of the exchange between SM and CM. The device (SM or CM) wishing to synchronize another device will send, at the time noted t_0 in its time base, a frame called Protocol Control Frames (PCF). The message is received by the receiver at time t_{rec}, after a known traversal time because the same route is always used, and therefore a constant propagation

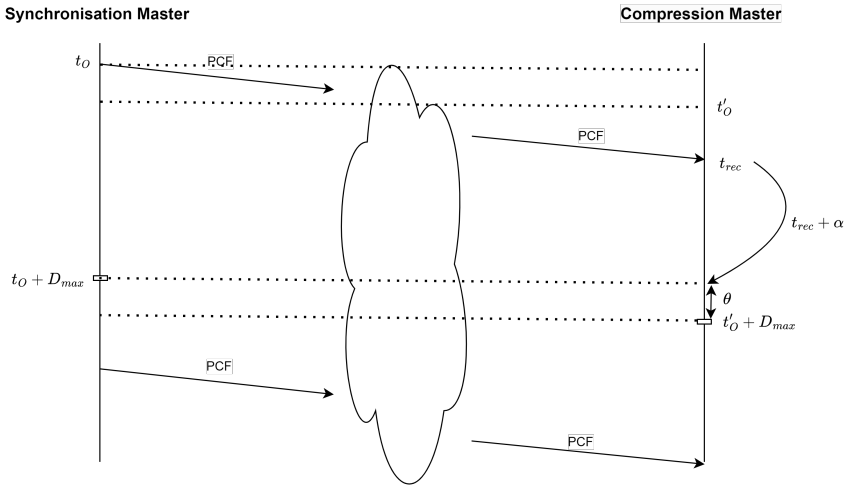


Figure 2.9: TTEthernet message exchange pattern between a Synchronisation Master and a Compression Master

delay, as well as a known residence time in the FIFOs of the output ports, as noted in the frame. At the moment of reception, a term noted α will allow to artificially delay the message, in order to place itself in conditions where this message would have undergone the greatest possible network latency. α is calculated using Equation (2.6) with T_{res} the residence time undergone by the PCF frame and $T_{res_{max}}$ the maximum residence time calculated during the network design phase.

$$\alpha = T_{res_{max}} - T_{res} \tag{2.6}$$

After this artificial delay, the CM knows that it should have received the message at $t'_0 + D_{max}$, where D_{max} is the sum of the propagation delay on the links and the maximum residence time $T_{res_{max}}$. The offset between the two clocks can thus be calculated using the following equation:

$$\theta = (t'_0 + D_{max}) - (t_{rec} + \alpha) \tag{2.7}$$

To the best of our knowledge, there is no public precision value that can be achieved by this protocol. Indeed, one of the limitations of this protocol is its closed nature, with hardware mainly marketed by TTTech, which is also the company behind the protocol and the main scientific publication on the subject of synchronization with it.

Another important element for a synchronization protocol intended for use on critical networks is robustness. SAE AS6802 meets this need, thanks to the convergence mechanism and by providing multiple SMs and CMs to withstand one or more SM or CM failures or link losses that render them inaccessible. In addition, compression also makes it possible to ignore SMs that send the wrong time.

Today, companies seem less and less interested in using TTEthernet for their new embedded networks, unlike TSN. But before the emergence of TSN, many critical embedded networks were seduced by the promise of a mixed-criticality network with high data rates achievable with TTEthernet. Examples include the Ariane 6 launcher, the Orion space capsule and the Lunar Gateway.

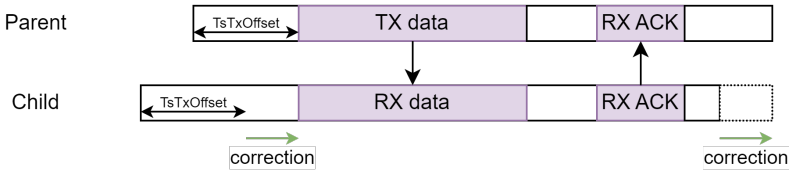


Figure 2.10: TSCH sync message exchange pattern

Wireless synchronization protocols

In the previous subsection, we focused on wired synchronization protocols, mainly using Ethernet. But as IEEE802.1AS can also be used for wireless synchronization, it's important to briefly introduce some wireless synchronization protocols.

Synchronization in the wireless world is mainly aimed at TDMA to avoid collisions, and at the niche of applications requiring localization which seems to be growing with this increasingly connected world. In recent years, the field has been driven mainly by the Internet Of Thing (IoT) and its constraints of low bandwidth, battery consumption and low clock quality (i.e. 20-100ppm) to reduce costs. So there are a number of very simple protocols.

In this field, Reference-Broadcast Synchronization (RBS) [53] is one of the first synchronization protocol. RBS build a common time base by convergence. It achieves millisecond precision by means of a periodic broadcast of a reference message. When the message is received by the different receivers, they note the time of arrival of the message and then exchange this record with each other. Using their readings and the exchanged readings of the last m reference message, the receivers compute an average of the m difference between the different reception times of the same reference message, in order to determine the clock offset of any other receiver. Thus, when exchanging with another receiver, it is possible to adapt to its time base.

Among the most modern and common is TSCH sync. TSCH's synchronization protocol achieves sub-millisecond precision to synchronize the start of transmit/receive slots. Synchronization is performed following the TSCH parent/child hierarchy. This does not require additional messages, but relies on a comparison of the reception time of a message with the actual time, as illustrated in Figure 2.10.

Several other protocols exist to meet different needs of this field. The most common are described by Djenouri et al. in [52].

2.5 IEEE802.1AS

Of the standards listed above, only PTP with a certain profile (like IEEE802.1AS), White Rabbit and TTEthernet offer the precision needed to control access to a high-speed medium in time. These three protocols carry synchronization in Ethernet frames, making it possible to synchronize switches or limited nodes not implementing an IP/UDP communication stack. However, TTEthernet is a closed standard, with few hardware components supporting it making them expensive. White Rabbit achieves a too low precision for embedded needs at the cost of dedicated hardware (but open-hardware), a more complex protocol (PTP + SyncE) and the obligation to use fiber optics. Whereas IEEE802.1AS is a simpler protocol, whose needs are covered on many network interface cards and, above all, is designed for use with other TSN protocols. In view of these advantages, we focus on IEEE802.1AS in the following. In this section we detail the protocol's history, how it works and the improvements currently under discussion in the IEEE TSN working group.

	Daytime Protocol and Time Protocol	NTP	PTP	SyncE	White Rabbit	TTEthernet
Targeted network	Arpanet	Internet	From small network to Internet	Telecom core network	Fiber Ethernet network	Critical embedded network
Synchronisation type	time	time	time	frequency	frequency + time	time
Hierarchic or convergence approach	hierarchic	hierarchic	hierarchic	hierarchic	hierarchic	convergence
Precision	few seconds	few milliseconds	milliseconds to sub-microsecond	< 50ns	sub-nanosecond	
Robustness to faillure		Multiple references and dynamic path of Internet	BMCA	Multiple references	BMCA	Multiple node with same role with a convergence mecanism

Table 2.1: Summary of the main features of synchronization protocols for wired networks. Note that precision varies greatly depending on the synchronization interval.

2.5.1 AS context and history

The first edition of IEEE802.1AS [22] was published in 2011 by the IEEE802.1 Audio Video Bridging (AVB) working group. The need for a precise synchronization protocol over Ethernet was driven by the use of AVTP, which enables multimedia streams to be played back at a precise moment on the receiver. If the multiple receivers share the same clock, then it's possible to play streams at the same time e.g. multiple loud speakers connected via Ethernet.

Not to reinvent the wheel, the AVB working group decided that AS would be a profile of IEEE1588-2008 [18], whose protocol is called Precise Timing Protocol (PTP). Adding constraints to the latter (such as the fact that any device in a synchronization domain must support AS) and choosing the mechanisms leading to the greatest precision (such as the peer-to-peer delay measurement) made it possible to fix the precision that AS had to reach to sub-microsecond at 7 hops.

The creation of the TSN working group with new industrial sectors has led to the creation of a new amendment to IEEE802.1Q to answer the need for hard real time scheduling : IEEE802.1Qbv [27]. This amendment, called Enhancements for Scheduled Traffic, proposes a mechanism allowing to divide in time slots the access to the Ethernet medium. This mechanism is called the Time-Aware Shaper (TAS) and relies on a network-wide common time to be used to its full potential. Indeed, although this shaper can use his internal clock (and time base) to schedule traffic, it is under efficient. Using a common time base, allows the network designer to define a global schedule for all the network exchanges. Thus a packet sent in a TAS window will be received at the other end of the link when its TAS window opens and can be transmitted without waiting. This mechanism makes it possible to greatly reduce the waiting time in the switches due to other higher priority messages and therefore reduces the latency and the jitter of the flow. It is particularly suited for real-time critical flows requiring low latency and/or jitter. With the addition of this new mechanism, the use of IEEE802.1AS, previously intended for distributed application synchronization, extends to a service for the network.

The new needs expressed by the new members of the TSN working group have led to a new version of IEEE802.1AS, named IEEE802.1AS-2020 [34] to meet the requirements of time-sensitive control. This new version brings mainly four new features:

- A way to use the protocol on 802.11 (WiFi) links and taking advantage of the propagation delay measurement mechanism that is part of the 802.11 standard: Fine timing measurement (FTM)
- Multiple domains that allows 1) the support of multiple time base (for example global and working clock in the factory automation world) and 2) fault tolerant and redundant synchronization.
- New ways to detect non-AS devices in the network in order to avoid them for a better precision
- A new mechanism from IEEE1588-2019 [32] and called Common Mean Link Delay Service.

At the time of writing this manuscript, this IEEE802.1AS-2020 is the current version of the IEEE802.1AS standard.

2.5.2 gPTP operation

Generalized Precision Time Protocol (gPTP) is the implementation of the IEEE802.1AS standard. Its operation is primarily based on two mechanisms :

- propagation delay measurement mechanism
- synchronization information distribution mechanism.

These two mechanisms are described in the first two subsections. Then other additional mechanisms are detailed.

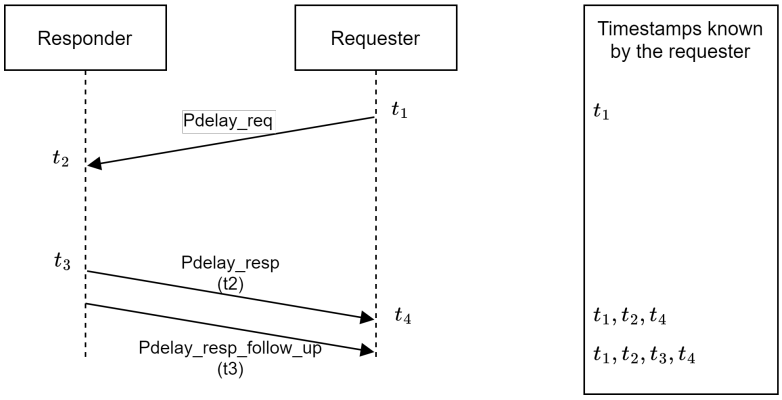


Figure 2.11: Illustration of a peer-to-peer delay mechanism.

Propagation delay measurement mechanism

In order to obtain a precise synchronization, the first step is to determine the delay that the synchronization message has spent on the link to take it into account when computing the clock correction to be applied. To do this, IEEE1588 offers two mechanisms: the end-to-end or the peer-to-peer propagation delay measurement mechanisms. To meet TSN's need for precision, the peer-to-peer mechanism was chosen thanks to its higher precision.

The peer-to-peer propagation delay measurement mechanism, often called *Pdelay* mechanism, is executed by every gPTP port. It measures, periodically the port-to-port propagation time, sometimes called the link delay, between two connected ports supporting the *Pdelay* mechanism. It uses three messages to generate and exchange four timestamps that are use to estimate the link delay. The complete exchange is depicted in Figure 2.11.

This exchange is triggered by a requester by sending a `Pdelay_req`. When sending this request, the requester will store t_1 ,the transmission timestamps of the message. The `Pdelay_req` is received by the responder at a timestamps noted t_2. The request also triggers the emission of a response message called `Pdelay_resp` which carries t_2. This message is sent (respectively received) at a timestamp denoted t_3 (respectively t_4). To carry t_3 to the requester, a new message, called `Pdelay_resp_follow_up`, is issued by the responder.

The use of a second message to transmit the transmission time of the previous AS message is called two-step mode. A one-step mode is also described in the standard but it requires special hardware that can write the transmission timestamp in the message while sending it, mitigating the need for a follow_up message. Due to the lack of industry interest in this mechanism, its lower accuracy and the discussions taking place in the IEEE802.1ASdm amendment working group to remove this mode, we will focus on the two-step mode for the rest of this manuscript.

From the four timestamps, to estimate the link delay D the protocol uses Equation (2.8). The result is often called *Pdelay*.

$$D = \frac{(t_2 - t_3) + nr(t_4 - t_1)}{2} \tag{2.8}$$

nr is the *neighborRateRatio*. It compensates the relative clock drift between the requester and the responder. It can be computed by Equation (2.9), using t_3 and t_4 timestamps from two consecutive peer-to-peer propagation delay measurements procedures. This mechanism is called logical syntonization because it allows to perform the calculations in a time base having the same frequency

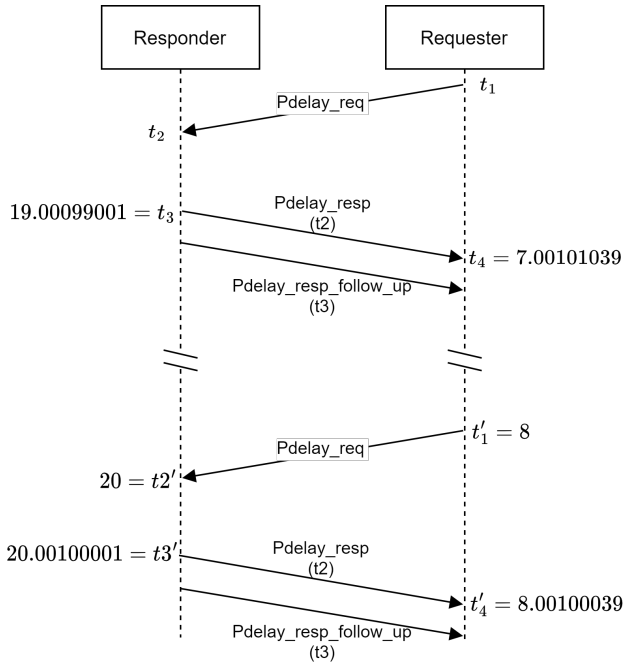


Figure 2.12: Two consecutive peer-to-peer delay mechanism with numerical values.

(syntonization) without the need for specialized hardware clock that can correct their frequency at the hardware level.

$$nr = \frac{f_{req}}{f_{resp}} = \frac{t'_3 - t_3}{t'_4 - t_4} \tag{2.9}$$

This mechanism is executed periodically following an interval called *pdelayInterval*. Default *pdelayInterval* is 1 second. These periodic executions may seem useless because in a wired Ethernet network the link delay will not evolve (unlike in wireless especially with mobility). However, the periodic execution makes it possible to keep up to date a *neighborRateRatio* which is used by the synchronization mechanism but also ensure that the port on the other side of the link is always able to support gPTP. The standard also recommends using a filter (without specifying the nature of this filter) to smooth out the inaccuracies of this mechanism. Thus obtaining the estimated delay periodically allows to be able to average it (or to use a more complex filter) with the previous values.

To illustrate what has been described in this subsection, Figure 2.12 provides the timestamps needed to instantiate the peer-to-peer delay propagation measurement mechanism as computed with Equation 2.10 and Equation 2.11.

$$nr = \frac{f_{req}}{f_{resp}} = \frac{20.00100001 - 19.00099001}{8.00100039 - 7.00101039} = 1.00002 \tag{2.10}$$

$$D = \frac{(t'_2 - t'_3) + nr(t'_4 - t'_1)}{2} = \frac{(20 - 20.00100001) + 1.00002 * (8.00100039 - 8)}{2} = 200ns \tag{2.11}$$

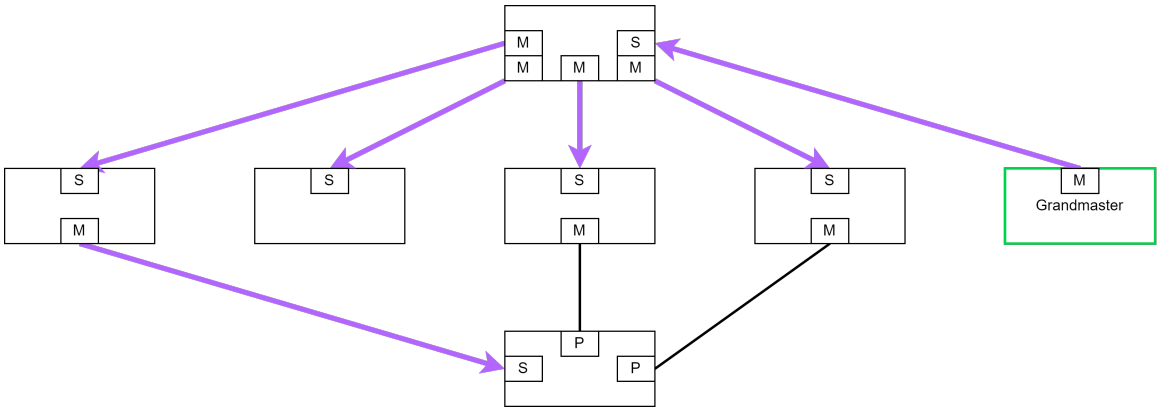


Figure 2.13: Illustration of one of the possible spanning tree to distribute the synchronization information on the core automotive topology

Synchronization information distribution mechanism

The second important element of a synchronization protocol is the distribution of the source time. The device which is used as the time source is called Grandmaster (GM). Its time can be from its internal clock oscillator or from an external source like a dedicated link, NTP, GPS or an atomic clock.

To synchronize clocks across a network, gPTP uses the master/slave paradigm. Such a network is depicted in Figure 2.13. It interconnects a set of time-aware systems (switches or end-nodes) using a spanning tree defined by the state of each port. The Grandmaster, the time-aware system with all its port to the master state, distributes its clock to the other devices, using a spanning tree. It broadcasts `Sync` and `Follow_Up` messages on its Master ports (M in Figure 2.13). Each time-aware system receiving these messages on a slave port (S in Figure 2.13) forwards it on its master ports. The passive ones (P in Figure 2.13) ignore them to avoid cyclic dependencies.

To be more specific, every *syncInterval* (by default every 125ms), the Grandmaster sends a `Sync` message out of its master ports, followed by a `Follow_Up` message containing O, the exact transmission time of the `Sync` message (named the *preciseOriginTimestamp* in the standard), as pictured in Figure 2.14. As for the `Pdelay_resp_follow_up`, `Follow_Up` is used for the two-step mode. `Sync` and `Follow_Up` are received via the slave ports of the time-aware system connected to the Grandmaster. If the receiving device has ports in the master state, it directly forwards the `Sync`. Next, it updates the `Follow_Up` message with the newly computed *rateRatio* r and *correctionField* C. Once updated, it forwards the latter to the next time-aware system.

The *rateRatio* r_i allows for logical syntonization of a time-aware system i to the Grandmaster rate. It is the product of the *neighborRateRatio* values calculated by the receiver ports on the path going from the Grandmaster to the time-aware system of interest. It is initialized to 1 by the Grandmaster and is updated on each hop using Equation (2.12), where i is the receiving node and $i-1$ the sending node.

$$r_i = r_{i-1} \times nr_i \tag{2.12}$$

The *correctionField* C carries the time elapsed in the time-aware systems and on the links on the path between the Grandmaster and the time-aware system preceding the last hop. At hop i, C_i is calculated using the previous correction field C_{i-1}, the previous *rateRatio* r_{i-1}, its current

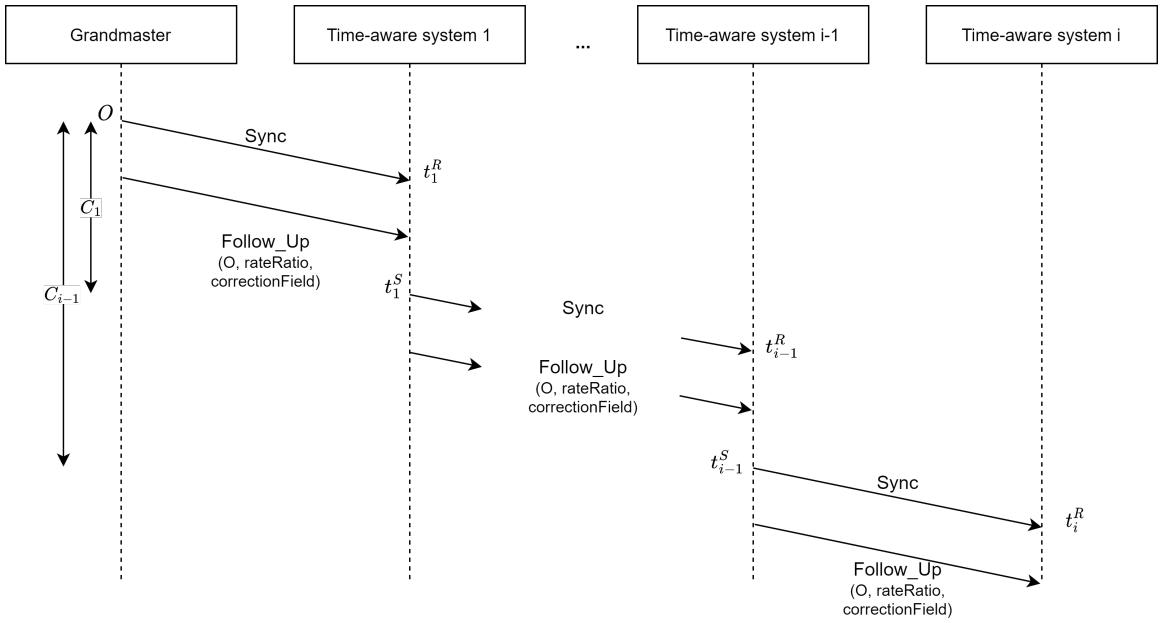


Figure 2.14: Illustration of the synchronization distribution mechanism in a i-hop network.

neighborRateRatio nr_i, its current value of D_i and the residence time $t_i^S - t_i^R$ of the `Sync` in its buffer following Eq. (2.13):

$$C_i = C_{i-1} + D_i \times r_{i-1} + (t_i^S - t_i^R) \times r_{i-1} \times nr_i \tag{2.13}$$

At each `Sync + Follow_Up` reception, a time-aware system i computes the correction to apply to this clock by calculating the difference between its local time and the estimated Grandmaster time $GM_i(t)$. This correction value can either be positive, if the time-aware system i is late, or negative, if the time-aware system i is ahead of the Grandmaster time.

The Grandmaster time $GM_i(t)$ is estimated by system i with Eq. (2.14):

$$GM_i(t) = O + C_{i-1} + D_i + (t - t_i^R) \tag{2.14}$$

where O is the original time of `Sync` transmission by the Grandmaster, C_{i-1} the correction field of the `Follow_Up` message, D_i the previous hop link delay retrieved by the peer-to-peer delay procedure and $(t - t_i^R)$ the time elapsed since the reception of the last `Sync`.

Note that the standard does not propose a method to correct the clock but only how to compute the offset to the Grandmaster time. It is therefore possible to find for example devices able to correct their clock very quickly and others correcting their clock slowly to avoid edge effects on their distributed application.

Figure 2.15 gives numerical values to illustrate the mechanisms presented in this subsection on two hops. On this example, we only detail the computation needed for the end-node synchronization but of course the switch does similar computation to synchronize itself. Let's assume that $D_1 = 200ns$, $nr_1 = 1.00002$ and $D_2 = 198ns$. When the switch forwards the `Sync` to the end-node, it computes $r1$ and C_1 as follows, in order to update the content of the `Follow_up`.

$$r_1 = r_0 \times nr_1 = 1 \times 1.00002 = 1.00002$$

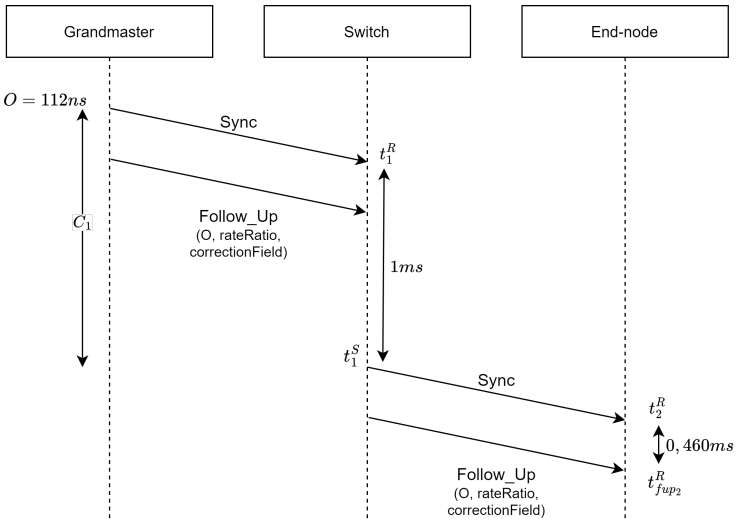


Figure 2.15: Illustration of the synchronization distribution mechanism in a 2-hop network with numerical values.

$$\begin{aligned} C_1 &= C_0 + D_1 \times r_0 + (t_1^S - t_1^R) \times r_0 \times nr_1 \\ &= 0 + 200 \times 10^{-9} \times 1 + 1 \times 10^{-3} \times 1.00002 = 1,00022ms \end{aligned}$$

Thus, when receiving the `Follow_up`, the end-node can estimate the Grandmaster time at the same instant, as illustrated with the following equation, in order to deduce an offset between its time and the estimated Grandmaster time that can be used to feed the clock correction algorithm.

$$\begin{aligned} GM_2(t_{fup_2}^R) &= O + C_1 + D_2 + (t_{fup_2}^R - t_2^R) \\ &= 112 \times 10^{-9} + 1,00022 \times 10^{-3} + 198 \times 10^{-9} + 0.460 \times 10^{-3} = 1,46053ms \end{aligned}$$

Synchronisation tree

The `Sync` and `Follow_Up` distribution spanning tree as well as the Grandmaster can be chosen dynamically, using the Best Master Clock Algorithm (BMCA), or statically.

Best Master Clock Algorithm The BMCA is a distributed algorithm that dynamically determines the state of the ports of each time-aware system and elects a Grandmaster. With this mechanism, every time-aware system that can be Grandmaster sends periodically (every 1s by default) `Announce` messages to inform other devices about its clock quality. `Announce` messages are broadcasted to every other devices that are connected to the `Announce` receiver. This message carries the following information about the sender clock :

- *grandmasterPriority1* : configurable priority to force a Grandmaster
- *grandmasterClockQuality* : value formed by the *clockClass* (denotes the traceability, synchronization state and expected performance of the time or frequency distributed), *clockAccuracy* (the expected time accuracy), and *offsetScaledLogVariance* (offset representation of an estimate of the clock variance)

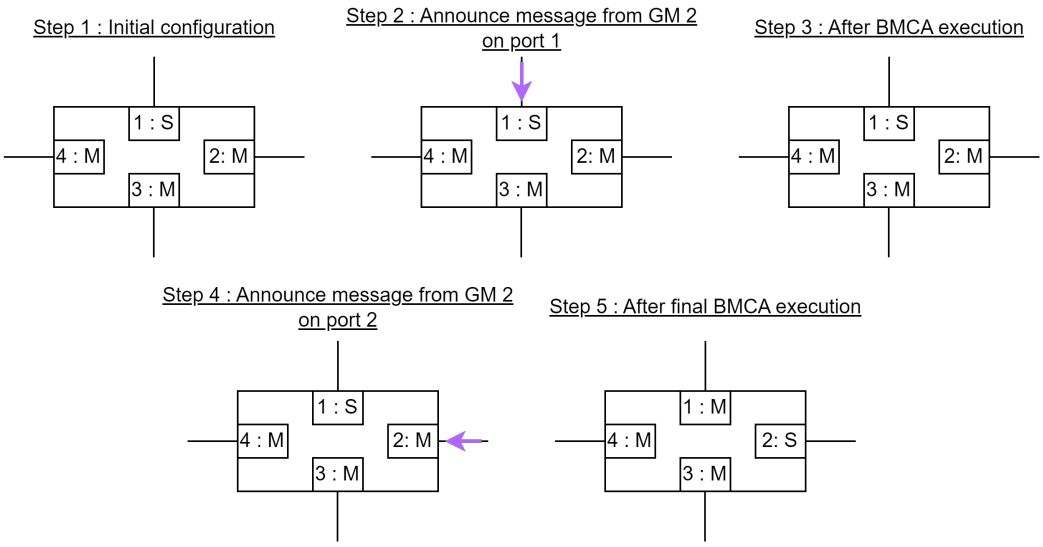


Figure 2.16: An example of a BMCA execution on a switch

- *grandmasterPriority2* : configurable priority to force a Grandmaster
- *grandmasterIdentity* : Extended Unique Identifier (EUI-64) of the clock
- *stepsRemoved* : number of hops that the `Announce` message has undergone
- *timeSource* : time source used by the sender such as Atomic clock, GPS, terrestrial radio, NTP, internal oscillator...

With this information, it is possible to execute the BMCA. Let's explain this execution with the help of the example shown in the Figure 2.16. Let's take the example of a 4 port switch whose ports are numbered from 1 to 4. Port 1 is in the slave state and receives `Sync` and `Follow_Up` messages from Grandmaster 1. The other ports are in the master state and forward these messages as pictured in step 1. In step 2, a new Grandmaster, GM 2, is connected to the network. This new Grandmaster, because of the internal execution of the BMCA, determines that it is better than GM 1, so it switches all his ports to master state and sends `Sync`, `Follow_Up` and `Announce` messages. Our illustration switch then receives `Announce` messages from the new Grandmaster through ports 1. It executes the BMCA to determine the new configuration of these ports. To determine the new Grandmaster, the BMCA compares a set of parameters of the Grandmaster currently used by the switch with the ones of the potential new Grandmaster described in the `Announce` message. The set of parameters is described in the following list.

- GM priority1
- GM class
- GM accuracy
- GM offset scaled log variance
- GM priority2
- GM identity

This comparison takes place in the same order as the one of list. Let's assume that our new Grandmaster has the same priority1 as the current GM but has a better clockClass. In the second

step of the comparison, the algorithm will determine that the candidate GM is better and will therefore set the port on which the `Announce` message was received to slave state and the others to master state. The message is received on port 1, there is no change in the status of the ports, but the switch will synchronize to the time base of the new GM and forward these `Sync` messages as depicted in step 3. When another `Announce` message from GM2 is received on port 2, as shown in step 4, the algorithm is executed again, but since the source of the `Announce` messages is identical, it is not possible to differentiate them with the previous parameters. In this case, the algorithm uses the StepRemoved value of the message to select the port that received the `Announce` with the least amount of hops as the port to elect as slave. In case of a tie, the ultimate tie breaker is the unique identifier of the port that sent the `Announce` message on the GM. In the latter case the losing port goes into the passive state. In our example port 2 is chosen as best by distance. So it becomes the slave and the other ports are in the master state as illustrated in step 5.

Of course, while the switch is executing the BMCA, it is also executed on the other time-aware systems that have received the `Announce` messages from the new Grandmaster.

External Port Configuration Much simpler than its dynamic equivalent, this mechanism makes it possible to define the state of the ports statically. Thus, the `Sync` distribution spanning tree and Grandmaster are given by the static configuration of all the gPTP ports. However, the standard does not provide a mechanism to distribute this static configuration but mention the possible use of implementation specific solution or the use of well know configuration protocol like SNMP or YANG-based protocol [55] [44].

Domains

A gPTP domain is a set of time-aware systems synchronized on the same Grandmaster. Each domain is identified by an id, which is stored in the gPTP message header. Each domain has its own spanning tree to distribute its synchronization information.

This mechanism enables the use of several domains to share different time bases, such as a working time base and a UTC base on the same network. But it can also provide redundancy in the distribution of the same time base. Indeed, when combined with External Port Configuration mechanism, having several domains makes it possible to have several paths for distributing synchronization information in the event of a link or time-aware system failure and several grandmasters in the event of a GM failure. However, some aspects of this robustness, such as domain switching in the event of a failure, have not yet been standardized and are currently being discussed in the P802.1ASdm amendment, which will be detailed in the following subsection 2.5.3.

The capabilities of this mechanism are illustrated on the Figure 2.17. Two grandmasters can be observed with three domains. Grandmaster 1 is the Grandmaster of the ruby domain and Grandmaster 2 of the green and purple domains. The single port shown in GM 2 is both master for the green and purple domains and slave for the ruby domain. We also observe that the green domain synchronizes only a part of the network.

Although multiplying the number of `Sync`/`Follow_Up` carries new information (the synchronization information of each domain), multiplying the number of *Pdelay* messages does nothing. Thus, with the addition of domains in the latest version of the standard, a new mechanism called Common Mean Link Delay Service (CMLDS) has been introduced. This new mechanism allow the protocol to measure the delay of the link only in one domain and pass the value of this delay to the other gPTP instance present on this port. If we go back to the example depicted in the Figure 2.17, the left port of the bottom switch executes the Pdelay mechanism on the purple domain and the gPTP

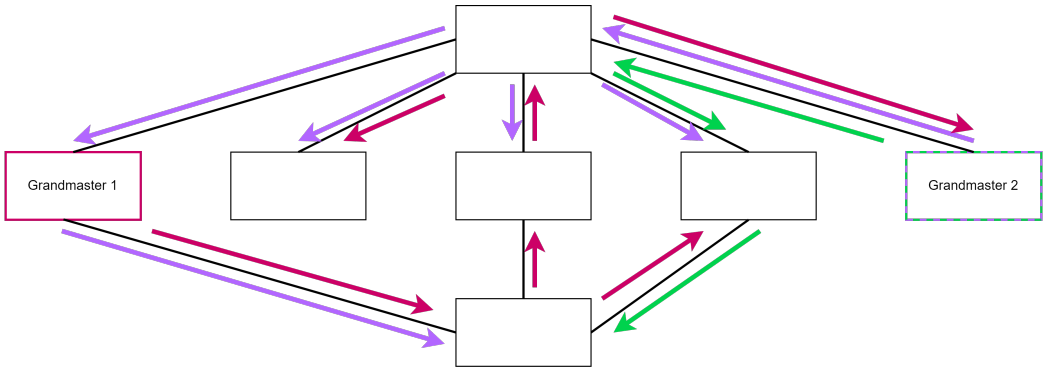


Figure 2.17: Illustration of possible configuration with three domains on the core automotive topology

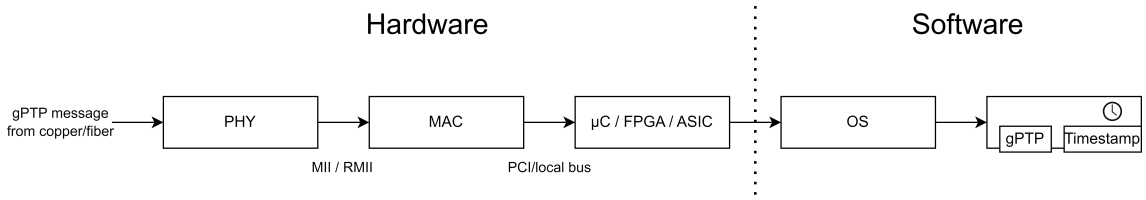


Figure 2.18: Illustration of an architecture that implements software timestamping

instance responsible for this port for the purple domain provides the result to the other instances present on the port. In this example, the only other instance present is the ruby instance.

Timestamping

As we will see later in this manuscript, synchronization precision depends greatly on timestamping accuracy. There are several ways to implement gPTP timestamping. In this section, we discuss the two main implementations.

Software timestamping The simplest way to implement a gPTP system is to perform timestamping in the gPTP stack within the operating system (OS) as illustrated on Figure 2.18. However, this software timestamping can lead to great inaccuracy. Indeed, between the reception (or the emission) of the message by PHY/MAC layer and the timestamping at the software level, a lot of jitter, caused by other applications or by the OS, can alter the timestamp precision.

A very simple experiment gives us a precision close to 70ms with software timestamping. This result is an average of 1000 measurements of precision between a Meinberg microSync HR (GPS clock) that act as Grandmaster and an I210 Network Interface Card installed on a desktop computer running stock Ubuntu OS. The computer uses the open source utility *ptp4l* to implement gPTP with the timestamping software mechanism. Precision is measured using a NetTimeLogic PPS analyzer. The two time-aware systems are connected by a 1GB/s link. During this experiment, measured precision was between 63ms and 72ms with a standard deviation of 2ms

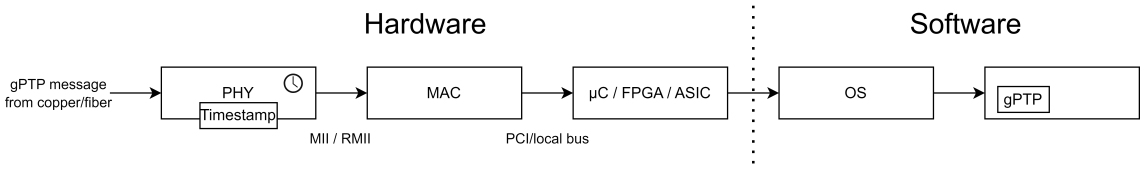


Figure 2.19: Illustration of an architecture that implement hardware timestamping

Hardware timestamping To bypass the software-related jitter, another mechanism exist : hardware timestamping. As described in the Fig 2.19, this mechanism relies on a timestamping step at one of the first two OSI layer (PHY or MAC). The timestamp then goes up the protocol stack to the gPTP application with the message. This mechanism then makes it possible to implement gPTP at any level in the network stack (software or hardware implementation of gPTP) without altering the precision of the timestamps. However to achieve this hardware timestamp, the PHY or MAC chipset must support this feature.

By reproducing the previous experiment and enabling hardware timestamping, we obtain a mean precision of 34ns between the two time-aware systems. In detail, measured precision was between 20ns and 54ns with a standard deviation of 4.7ns.

Due to the need of high precision synchronization and thanks to the profusion of chipset supporting Hardware timestamping (even low cost and mainstream ones like Intel I2XX family), the deployment of a network supporting gPTP relying on hardware timestamping is a reasonable assumption.

Message

The messages used by gPTP can be divided into two categories: event and general messages. The event messages are messages that need timestamping at the emission and the reception. These messages are `Sync`, `Pdelay_Req` and `Pdelay_Resp` messages. The remaining messages (`Pdelay_Resp_Follow_Up`, `Follow_Up`, `Announce` and signaling messages) are part of the general message class and have to be treated as normal messages.

Signaling messages are another type of messages provided by the standard. These messages carry information, requests, and/or commands between two gPTP ports that are connected to the same link, via one or more Type Length Value (TLV) fields. In IEEE802.1AS-2020, a signaling message can be use to request that the port at the other end of the link sends `Sync` messages, link delay measurement messages, or Announce messages at desired intervals; to indicate whether the port at the other end of the link should compute neighborRateRatio and/or link delay; and to indicate whether a PTP Port can receive and correctly process one-step `Syncs`. One usage of this functionality is to allow a time-aware end-node in power-saving mode to ask for a reduction of the *syncInterval* and *pdelayInterval* to his switch. However, on the three commercial implementations that we consider in this thesis, none of them supports this message type. They seem to be less of a priority for silicon vendors than the core mechanisms described above.

Among the options proposed by IEEE1588, it was chosen to transport the gPTP messages in Ethernet 802.3 frames. These Ethernet frames must respect the following rules:

- shall not have a VLAN tag nor a priority tag
- The EtherType of `Sync`, `Follow_Up`, `Pdelay_Req`, `Pdelay_Resp`, and `Pdelay_Resp_Follow_Up` shall be 88-F7

- The destination mac address of `Sync`, `Follow_Up`, `Pdelay_Req`, `Pdelay_Resp`, and `Pdelay_Resp_Follow_Up` shall be 01-80-C2-00-00-0E. This address is a multicast reserved address for gPTP

The use of other physical layers such as WiFi or EPON are also described in the standard but will not be covered in this manuscript as they are not used in the critical embedded world.

The gPTP messages are composed of a common header which is completed by a body specific to each message type and additional TLVs if needed. To understand the following chapters, it is not necessary to knows in detail the content of gPTP messages. However, it is important to underline for the remainder of the manuscript that the gPTP messages are very small. An Ethernet frame containing a `Sync` is made of 64 bytes, 94 bytes for a `Follow_Up` and 72 bytes for each Pdelay message.

2.5.3 Future of AS standard

At the time of writing this manuscript, the TSN working group has four open Project Authorization Requests (PARs) to amend 802.1AS-2020. The objective of these amendments are detailed in the following subsections

P802.1ASdn : YANG Data model

P802.1ASdn aims to specify a YANG data model that allows configuring and state reporting for all managed objects of the base standard IEEE802.1AS. YANG is a data modeling language developed by the IETF. A YANG model defines a hierarchical data structure, which can be used for operations based on network configuration management protocols (such as NETCONF/RESTCONF). The operations include configuration, status data, remote procedure calls (RPCs), and notifications.

Similar amendments are also being discussed for other TSN standards to provide standardized means of configuration and monitoring for all TSN devices. This amendment illustrates the shift of the industry from MIB/SNMP to YANG/NETCONF (or other YANG-based protocol) due to the limitations of SNMP in terms of device configuration as explained in [47].

P802.1ASdm : Hot standby full support

IEEE802.1AS-2020 adds to the previous version, IEEE802.1AS-2011, the possibility to use multiple gPTP domains and static port state configuration. These two new mechanisms combined can allow a static and robust synchronization but, due to lack of time, a certain number of points allowing the operation of these mechanisms providing robustness were excluded from the base standard to be treated in an amendment: P802.1ASdm. Thus, this amendment aims to standardize the missing functions of these mechanisms like :

- Domain selection among the domains available for synchronous network applications and mechanisms such as TAS.
- Domains quality evaluation
- Hot standby Grandmaster architecture and behavior

A Hot Standby Grandmaster would have the ability to be a slave in a first domain and thus be synchronized with the GM of this domain, while being a Grandmaster in a second domain in order to broadcast the time base of the first domain.

This Grandmaster proxy is very promising in a static context where the robustness of the synchronization is critical. Indeed, such system could offer a more robust synchronization to one or

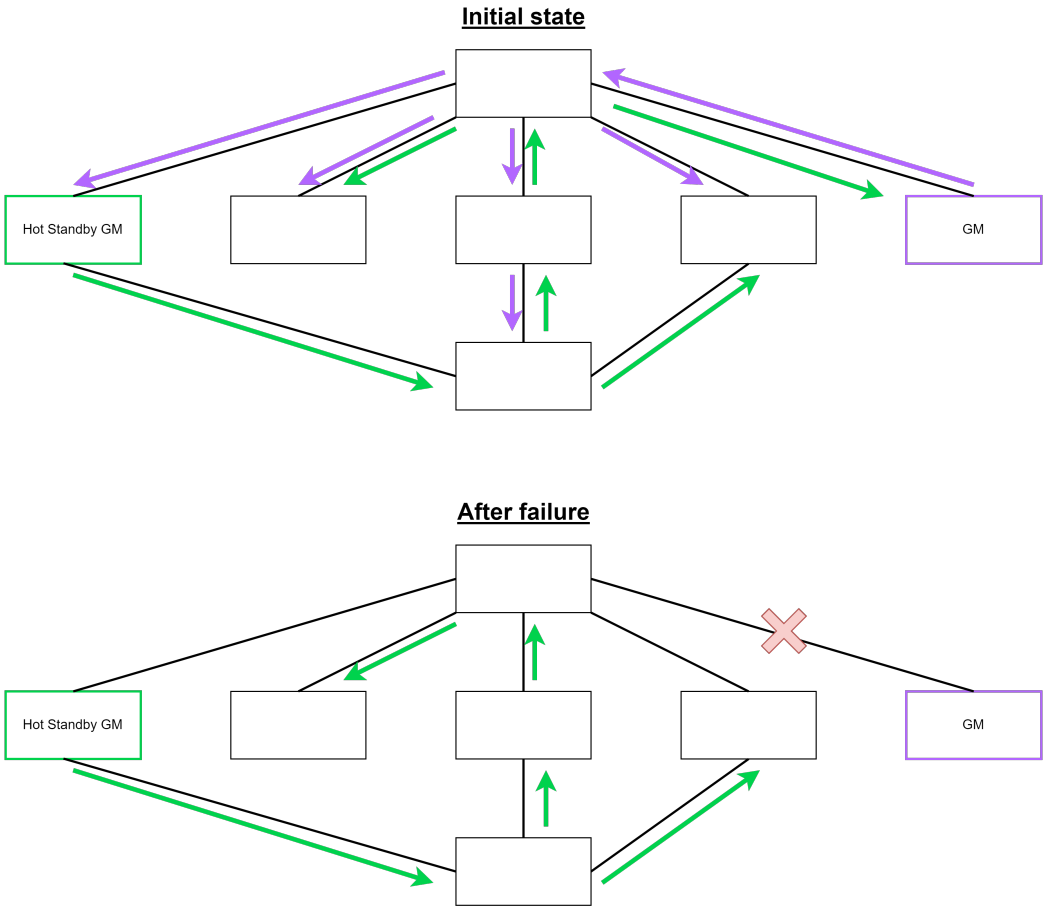


Figure 2.20: Illustration of Hot Standby mechanism on the core automotive topology

multiple failures affecting Grandmasters, because of the existence of several Grandmasters as illustrated in Figure 2.20. In this figure, in the event of a loss of communication with the GM, which had access to GPS time, all time-aware systems will lose the purple domain. However the Hot standby GM can take over without having access to an external time base, but only by maintaining the time base of the old GM in free-running with the green domain. Such a time base will then drift away from the GPS time base but, provided the Hot Standby GM's oscillator is properly chosen, can remain close enough to this time base to respect the synchronization specification until the end of the system's mission e.g. plane landing or safe car stop. Such a time base also makes it possible to keep a common time base for the network and distributed applications, even if it's not identical to the external time base.

Moreover, it allows to limit the time jumps during GM changes because the hot standby GM are only repeaters of the same time base. Indeed, having two Grandmasters, each with a different external source, e.g. GPS and NTP, could lead to a time jump of a few milliseconds when switching from the GPS Grandmaster to the NTP Grandmaster after a GPS Grandmaster failure. Such a jump could then greatly disrupt critical synchronous applications or critical network message distribution based on the use of TAS.

Finally, this mechanism should also offer an alternative to the multiplication of GMs for redundancy that can be limited by system constraints (e.g. we can't put a GPS antenna on a device that would be in the middle of an airplane cabin) or economic constraints (the cost of a precise grandmaster is important) by offering a relatively simple solution to implement.

802.1ASdr : Inclusive terminology

802.1ASdr is a corrigendum that aims to change the non-inclusive terms present in the base standard. This amendment is the consequence of a new rule of the IEEE Standard Association (IEEE SA) which indicates that "IEEE standards should be written in such a way as to avoid non-inclusive and insensitive terminology". Thus, the use of non-inclusive terms such as "master" and "slave" to describe port states and clock roles in a gPTP network is no longer allowed in IEEE standards. IEEE P1588g is also developing a consensus on the preferred alternative terminology. In order to avoid confusion in industry, 802.1ASdr will use the terminology determined by IEEE P1588g to describe PTP functionality.

The alternative terminology being still under discussion by the PTP working group, we will use the non-inclusive terms in the rest of this manuscript to avoid inconsistencies with literature and current standard version (IEEE1588-2019 and 802.1AS-2020).

P802.1ASds : Integration of field bus technologies

P802.1ASds specifies protocols, procedures, and managed objects that support Media Access Control (MAC) operating in half-duplex. This amendment is driven by the standardization of a new physical layer which is of great interest to the industry. This new physical layer, named 10Base-T1S, is a 10Mb/s fieldbus on a single twisted pair with an algorithm to avoid collisions, called PLCA. However, IEEE 802.1AS-2020 does not work on this bus (or any other buses) due to a issue known as "pdelay issue".

2.5.4 Deployment status

On a real-world level, IEEE802.1AS has been used in the world of professional audio/video for around ten years. Examples in the audio world include switches and input/output interfaces from MOTU, and mixing consoles from Presonus featuring Ethernet AVB ports. This equipment implements Credit-Based Shaper to guarantee bandwidth for each audio channel, and gPTP to synchronize the clocks used for audio sampling on the various devices.

In the world of factory automation, gPTP is also beginning to be used on new machines. The reason for this is the need to use the Time-Aware Shaper to keep Time Trigger operation similar to historical technology in this field. In this context, the extension of gPTP to the wireless world of the factories of the future is currently under study.

As far as critical embedded systems are concerned, the automotive industry is ahead of the aeronautic and space industries in the TSN field, thanks to its involvement from the very start of the TSN working group. The Autosar and Autosar Adaptive standards include descriptions of APIs specific to gPTP, and AVB networks (CBS + gPTP) are present in new cars for low criticality functions. The subject of the TSN used to meet the needs of critical systems is currently a research subject.

For the aeronautic and space industries, the need for mature, reliable technology combined with validation/certification constraints is today forcing TSN and gPTP into only the R&D domain especially for high criticality systems. In these industries, the first commercial use of TSN is scheduled

for 2026 in the cabin (low criticality systems) of A320 family aircraft, as part of AIRBUS's intelligent Core Management Platform.

Chapter 3

PhD problem statement

Driven by the industrial and scientific community's interest in TSN, a number of works have been carried out to study IEEE802.1AS, which is one of its major building blocks. These include works such as [89], [85] and [43] which address the problem of using gPTP in WiFi or 5G to unify wired and wireless communication in the factory of the future. Other works such as [71] and [48] have quantified the benefits of a synchronization protocol in terms of guaranteeing latency for other flows in other real-time networks. But in this chapter, we focus only on works studying IEEE802.1AS with the objective of either improving synchronization precision or its robustness. Some exceptions take place when works on PTP also apply to gPTP. This chapter is organized into a first section presenting scientific contributions whose aim is to improve synchronisation precision and a second section presenting contributions whose aim is to guaranty or improve robustness. The third section underlines the shortcomings of previous works for the deployment of gPTP in a critical embedded network. From this analysis, we draw conclusions on the necessary steps to offer a precise and robust gPTP deployment and motivating the core contributions of this thesis.

3.1 gPTP precision related work

In the literature, there are three ways to study the precision achievable with gPTP. The first one is simulation. It is an excellent way of refining the understanding of a protocol by adding measurement points that are either not possible or very difficult to implement in reality. Simulation also enables us to explore a vast configuration space, avoiding the limitations of real-life implementations, and to easily test new mechanisms. The second method is experimental study. This step is essential to study how the protocol actually works and to validate the representativeness of the simulated model. And finally, formal analysis. Like simulation, formal analysis relies on modelling the protocol to ensure that it guarantees its properties. As discussed in chapter 1, this step is mandatory in the critical embedded context to provide the proof needed for validation/certification of the system.

Simulation In order to study the precision achievable with gPTP, simulation is a tool that avoids researchers and engineers to depend on specific hardware implementations and lets them explore a vast design space of new mechanisms and/or configurations very easily. Work on this subject began very early in the standardization process. Indeed, in 2009, Garner et al. in [61], which is the editor of IEEE802.1AS, presented the progress of the draft standard produced by the Audio Video Bridging (AVB) working group, as well as simulation results. At the time, IEEE802.1AS

was intended to meet the needs of four use cases envisaged for the protocol: residential audio/video (A/V), professional A/V, industrial automation, and automotive. In its simulation part, this paper focuses on meeting the needs of the residential and professional A/V use cases. The simulator used is based on discrete-event simulation. The events modelled are the sending and receiving of `Sync`, `Pdelay_Req` and `Pdelay_Resp`, i.e. one-step operation. The protocol configuration values are the default values, sending `Sync` every 0.125s and `Pdelay_Req` every second. Oscillator drift is not modelled, and propagation delay is assumed to be constant and symmetrical. The study takes place on a daisy chain of 7 hops and lasts 10 seconds. The results of the simulations confirm that gPTP's performance can meet the need for precision and precision variation of different A/V standards such as residential and professional audio, as well as video with HDTV and uncompressed SDTV.

Lim et al. in [72] designed a simulation model based on OMNeT++[37] and INET [36] to study the use of IEEE802.1AS in an automotive network. This model implements a clock model with drift, granularity and initialisation time that can be specified differently for each network device. They study the peer-to-peer propagation delay measurement mechanism and the synchronization process in a 7-hop daisy chain network and in an automotive network. Using their model, they show that the synchronization is hardly impacted by the network load and highlighted the importance of the choice of the *Pdelay* averaging algorithm. They also conclude that their model is representative because their result are close to the ones predicted by the IEEE802.1 working group.

Diarra et al. in [51] used simulations to study the time taken to set up the common time base when a car is started up and the network needs a correct precision to distribute time-sensitive message. The simulator used is based on OMNeT++ and INET. This work is motivated by the need for synchronization to be established in 100ms, as for CAN and Flexray. However, gPTP's default configuration does not meet this need. The authors propose three ways of reducing this time. The first is to use a `Sync` and a `Pdelay_Resp` in order to have two messages to compute the *rateRatio* and thus not wait for the reception of a second `Sync` 125ms later. The second method consists in using the one-step mode in order to not have to wait for the reception of the `Follow_Up` and `Pdealy_Resp_Follow_Up` to continue protocol execution. The last proposal save time by storing the propagation delay in the device's flash memory. This value can come from the last execution of the propagation delay measurement mechanism before the vehicle was switched off, or from a predefined offline value. The results obtained on a three-hop network show that the optimizations as a whole enable the start-up time for the most distant device to be reduced from almost 400ms to less than 40ms, without any major deterioration in precision.

Gutiérrez et al. in [62] also developed a simulation library in order to study the distribution of precision as function of the time and the position of the switch in the network. Their study focuses on very large industrial networks, where the number of hops between the Grandmaster and the most distant device can reach 100 hops. The library, based on OMNeT++ and INET, implements, in addition to the main protocol functions, a clock model whose drift varies as a function of time at a configurable bounded rate, as well as an imprecision model impacting the accuracy of timestamps due to the physical layer (from [74]). However, the logical syntonization step is implemented using information (such as drift) that is accessible in the simulated environment but not in reality, inducing a gap with a test-bed deployment. Realistic sources of imprecision therefore have no effect on syntonization. Using the results obtained with the library, the authors deduce for each device a probability of being synchronized below a variable threshold. The further the device is from the Grandmaster, the lower the probability. For example, for a threshold of 1µs, the probability is 100% for the device located 20 hops from the source, compared with 85% for the device located 80 hops away.

Two groups of authors, Wallner et al. [90] and Puttnies et al. [81], have opted for publishing open-source simulation software. Both are based on OMNeT++ and INET, but differ in terms of

protocol. The first one implements PTP, while the second implements gPTP. Although the first library is not directly designed to simulate gPTP, its completeness means it can be used to study it. Indeed, it implements many of the mechanisms described in the PTP standard. It is therefore possible, for example, to choose to use the peer-to-peer propagation delay measurement to comply with the IEEE802.1AS profile. As well as being very complete from a protocol point of view, this library implements realistic clocks. Using another library called LibPLN also implemented by the authors, a noise composed of five colored noises whose predominance is configurable is added to the drift to precisely reproduce clock noise. As for the second open source alternative, it implements a much simpler clock model whose drift undergoes a linear evolution as a function of time. It also implements the basic functions of the gPTP, such as synchronization information distribution by `Sync` and peer-to-peer propagation delay measurement. As BMCA is not implemented, port status is statically defined using a configuration file. Logical syntonization is another missing mechanism in the implementation. Obtained results have been compared with the simulation results of Lim et al. [72] to confirm correct operation.

Except for the Gutiérrez et al. library, most of these libraries do not implement any source of imprecision other than the clock drift. Sometimes, clock granularity is added to clock drift to make the simulation more realistic. However, as we'll see in the next paragraph, other sources of imprecision can alter the quality of synchronization on real devices. But, these sources are only implemented in the work of Gutiérrez et al. which is unfortunately not open source.

Experimentation The precision aspect was also studied through experimentation on gPTP-supported hardware. As for simulation, one of the first experimental works was carried out by Garner et al. in [66] and [61]. In the first paper, the authors present the new features of the first draft version of IEEE802.1AS, as well as a presentation of a test-bed for evaluating the protocol. The test bench consists of a Grandmaster and n switches connected in a daisy chain. The various devices are equipped with an analog output enabling the offset of their respective clocks to be measured using an oscilloscope or dedicated instrument. Traffic generators can also be connected to the various switches to load the network in order to make it more representative of reality. The results obtained with this bench are presented in [61]. They observed a decrease in precision with the number of hops due to error accumulation. They measured periodic clock phase jumps of the order of 4ns, which they explain by the impact of the 8ns granularity of the device clocks on the propagation delay measurement. The authors also observe that, thanks to the higher priority of gPTP messages, other types of traffic have a negligible impact on the synchronization process. However, due to the limitations of the oscilloscope, the precision measurements obtained with the latter are only performed over a 2s window which may not be long enough to capture all variability.

In [83], Sakaguchi et al. implement gPTP in hardware on an FPGA. In a 1GB/s fiber-optic network, the worst-case precision measured with their implementation is 25ns after 3 hops, which is much better than the several hundred nanoseconds for software implementations of gPTP with hardware timestamping.

In the PTP-focused literature, other papers obtain results that are applicable to gPTP. The first of these papers is [67] by Robert M. Kaminsky who studies the short (15min) and long term (24h) holdover capacity in case of missing `Sync` and `Follow_Up`. The purpose of this study is to verify that the performance recommended by ITU-T in G.8273.2 can be achieved. The author tests the capabilities of three crystal oscillators: a TCXO, an OCXO and a DOCXO (Double Oven Controlled Crystal Oscillator). The results show that only the DOCXO achieves the performance required to remain in holdover mode for 24 hours. The TCXO fails after 20 minutes, compared with 2h30 for the OCXO. For the short term, only TCXO fails to meet the ITU-T recommendation.

In [63], Horita et al. use experimental measurements to study how to improve synchronisation

precision within a limited budget. The context of this paper is the telecommunications sector where replacing all network devices with PTP-supported devices is not an option because of its cost. The aim of the study is to understand how to improve precision by replacing only a few devices on the `Sync` path and by experimenting with the VLAN priority of PTP messages. For the experimental part, the authors ensure that measurements are reproducible by using available devices and, above all, by relying on test methodologies standardized by ITU-T G.8261 [16]. On the 6-hop daisy chain network with a network load of 80% of link speed, the reference measurement shows an approximate precision of 1µs. By segregating PTP traffic from the rest of the traffic using VLAN priority, an improvement in precision of 150ns is observed. Partial use of PTP-enabled devices on the path achieves a precision of 150ns in the absence of traffic, but deteriorates to 6µs when network load reaches 80%. With full deployment, precision is of 120ns and remains stable when network load increases.

PTP and gPTP rely on numerous timestamps. Imprecise timestamps imply imprecise synchronization. Starting from this premise, Loschmidt et al. in [74] investigate what makes hardware timestamping mechanisms imprecise in order to propose solutions for improving the synchronization accuracy of measuring instruments. Among the sources of imprecision described in the article are classic ones such as clock drift and granularity, or the time between two re-synchronizations. But it also describes a new source linked to the physical layer. Indeed, the latter can cause a random variation in the delay between the message's entry into the sender's physical layer and its exit from the receiver's physical layer. In addition to this random variation, constant asymmetry can also impact the accuracy of the propagation delay measurement mechanism, as well as the `Sync` travel time. In his thesis manuscript [73], Loschmidt presents, in a more comprehensive way, the results of his imprecision characterization for different physical layers. He points out that the form of imprecision is specific to each physical layer, and its dimension is specific to each implementation. He measures the best precision with the 1000Base-T physical layer, followed by 100Base-T and then 10Base-T with the physical interfaces at his disposal.

Several studies quantify the achievable precision as a function of different parameters. However, as precision is specific to each physical layer and implementation of that physical layer, it is very difficult to compare these different results with each other.

Formal methods In the world of critical embedded systems, formal proofs are regularly used to dimension systems for the worst-case scenario in order to avoid unforeseen situations. On the systems side, we find Worst-Case Execution Time (WCET) analysis. On the network side, Worst-Case Traversal Time (WCTT) is calculated using for instance Network Calculus analysis. As for the worst-case precision achievable with IEEE802.1AS, two papers [62][38] have studied it. In [38], Asokan et al. model the behavior of gPTP using the UPPAAL model checker to verify that the precision of all network devices is bounded. However, the bound on precision is not realistic, due to the lack of modeling of imprecision sources such as port-to-port propagation time variation or clock granularity. These sources of imprecision are modeled in the work of Gutiérrez et al. In addition to presenting simulation results, this paper presents a mathematical model for determining an upper bound on worst-case accuracy in a 100Base-T network. This model is based on an inventory of all the sources of imprecision that can act at each stage, in order to compute the maximum overestimation of the different mechanisms and deduce the maximum error. For example, the propagation delay measurement mechanism is impacted by clock drift between sending and receiving, clock granularity when taking timestamps, propagation delay variation due to jitter caused by the physical layer (from Loschmidt et al. in [74]). This imprecision can be compensated for on average, but in the worst case it can cause significant propagation delay variations. Gutiérrez et al. estimate that the propagation delay can be overestimated by 32ns at each hop. Combined with the imprecision that affects the

`Sync` journey, the model determines a bound on precision of 6.95µs after 100 hops. However, this work lacks of comparative studies with hardware supporting TSN to validate the model.

3.2 gPTP robustness related work

The literature defines two types of robustness which will be detailed in this section: resilience to failure and resilience to deliberate attack.

Failure resilience Link or node failures are inevitable, but in embedded systems they have to be contained and circumvented with a controlled impact on the rest of the network. Work on this subject has been carried out in related but equally critical fields such as electrical grid control. In [80], Puhm et al. study the possibility of using a second synchronization distribution tree with the BMCA using two protocols described in [25]. Parallel Redundancy Protocol (PRP) is based on the parallel use of a second network. High-availability Seamless Redundancy (HSR) uses the two directions of a ring topology to provide two redundant paths. Both protocols are well suited to message transmission, but from the gPTP point of view only one of the two paths will be selected by the BMCA. To get around this, the authors propose to synchronize two clocks (one per path) and select which one to use using a Kalman filter. Simulation results show that reconfiguration after a failure on one of the paths is much faster than with BMCA execution. In [68], Kyriakakis et al. adopt a similar approach but generalize it to more paths. Their experiments test 4 paths. Instead of using a Kalman filter, they study the use of a simple average and a Fault Tolerant averaging algorithm (FTA) to converge on an average clock from several clocks. The results show better synchronization stability with both filters than with BMCA. These two works only deal with BMCA, which is a complex algorithm compared to critical embedded practice. Static configuration could be a suitable alternative to BMCA for critical embedded networks, but to the best of our knowledge, there is no work on this subject.

In addition to link or node failures, the Grandmaster can also send erroneous information. Such failures are called Byzantine failures. This point was studied by Ferrari et al. in [57] and Dalmas et al. [49]. Ferrari et al. use static estimators to detect non-coherent variations in the Grandmaster clock. They show that these estimators enable a better assessment of the quality of synchronization, as well as detecting faults causing disconnection of the Grandmaster and faults causing erroneous time distribution. Dalmas et al. propose to use backup Grandmasters to monitor the current Grandmaster. Such Grandmasters act as slaves and calculate their offset relative to the current Grandmaster. If the offset exceeds a threshold, the backup Grandmaster alerts the current Grandmaster. If it receives enough alerts, the GM switches to Faulty mode and another Grandmaster take over using the BMCA. Experimental tests confirm the ability to detect and remedy this fault using the proposed changes.

Security When it comes to robustness, it's essential to study the impact of malicious behaviour. From a security point of view, a synchronization protocol is a very interesting target. Indeed, taking control of time makes it possible to control or disrupt synchronous distributed applications and network services that rely on sharing a common time base, such as TAS. Historically, PTP and its profiles, which can be found in the open world such as in the telecommunications sector or for factory automation, were much more widely studied than gPTP because they were more accessible than a protocol designed to operate in a closed network. As part of PTP, [60], [88] and [87] experimented with the following attacks:

- Rogue master : an attacker pretends to be a Grandmaster better than the current Grandmaster in order to be elected by the BMCA and control time on the network.
- Packet manipulation : a man-in-the-middle attacker falsifies data carried by legitimate packets.
- Packet delay manipulation : a man-in-the-middle attacker delays legitimate packets.

But today, with embedded networks opening up to the open world for ever more connectivity, the interest in studying gPTP has re-emerged. For example, in [58], Fotouhi et al. present a spoofing attack using gPTP. This consists in usurping a legitimate identity in order to propagate a false time base, desynchronize nodes by sending random timestamps, or perform denial of service. The authors show that these attacks can be carried out on a test bench consisting of two switches, a grandmaster, a slave and an attacker. They recommend the following solutions to limit these attacks:

- use of gPTP static configuration to prevent BMCA-based rogue master attacks
- use of multiple paths to detect attacks
- use of IEEE 802.1AE [31] to encrypt the payload content of MAC frames
- use of the optional TLV described in IEEE1588 to authenticate the message source

In order to avoid these attacks, work has been done to propose protection mechanisms. In [45], Buscemi et al. study the possibility of using dedicated devices called Intrusion Detection Systems (IDS) to detect a rogue master when BMCA is not available. An IDS monitors the network for suspicious traffic and alerts when it is discovered. The attack considered in the paper is a rogue master that adds an offset of $x$ns to each `Sync` transmission in order to progressively shift devices to a false time base. The proposed IDS monitors the clock behavior of the device on which it is implemented. It achieves a detection ratio of over 99% for offsets greater than 100ns. Smaller offsets are very difficult to detect, due to their low impact on the clock. The authors acknowledge that for detection of other types of attack, the IDS should also monitor incoming gPTP traffic.

3.3 PhD problem statement

As a reminder, our objective is to propose a robust and precise synchronization solution to meet the needs of critical embedded TSN networks. Our work is motivated by the strong industrial interest in TSN. We have seen that the IEEE802.1AS protocol is well suited to this need, thanks to its TSN-related design. However, during the literature review in the previous sections, several observations stand out. These observations have enabled us to identify the locks that need to be lifted in order to meet the need for robust and precise synchronization in critical embedded TSN networks. These locks led us to carry out the work presented in the remainder of this manuscript. They are presented in this section, starting with precision, then robustness, and finally the impact of the first two points on the rest of network traffic. These three subsections present the three main parts of the manuscript.

3.3.1 A precise synchronization

In a critical embedded network, the need for precision is highly variable depending on the industrial requirements, i.e. under microseconds for the case of satellite and milliseconds for the case of digital audio in the cockpit. However, it is important that this precision is bounded to dimension embedded systems and network scheduling in the case of Time-Aware Shaper use, but also for certification purposes.

In the literature, we note that several ways of assessing the precision of IEEE802.1AS stand out: simulation, experimental measurements and formal methods. All the work on simulation and formal

methods raises the question of representativeness in relation to reality. Indeed, no comparison has been made with real devices implementing gPTP. On the other hand, experimental measurements show highly variable results depending on the device and physical layer used, ranging from 25ns after three hops with fiber optics and a hardware implementation of the protocol to nearly 1µs using commercially-available switches after 6 hops and a network load of 80%. Work proposing a formal method for calculating a bound on worst-case precision is also very limited, with only one paper on the subject to the best of our knowledge.

We will take advantage of the growing availability of hardware supporting TSN to focus on the representativeness of the hardware in simulation tools and formal methods. Thus, for Part II of this manuscript, the objectives are i) to study the representativeness of a simulation tool from real measurements and ii) to propose a formal method for deriving a tight bound on the worst-case precision to meet the validation/certification needs of the critical embedded world thanks to fine modeling of inaccuracies sources. These two points are intended to provide the keys to understand and explore the parameters that impact precision, thus helping to explain the variable results obtained in experimental measurements. Moreover, the formal bound on precision is required to provide a robust deployment. Failure in the network will trigger a re-configuration of the network. This reconfiguration takes some time, that is ideally deterministic and known in advance. During this reconfiguration time, clocks are drifting away. Knowing a tight bound on precision can be leveraged to determine a maximum time for network re-configuration knowing clock's quality.

3.3.2 A robust synchronization

Regarding robustness, the literature distinguishes between robustness to failure and robustness to malicious attacks.

The aspect of robustness to malicious attack will not be dealt with in the remainder of this manuscript. Indeed, the literature seems to indicate that most security flaws could be contained if there wasn't a clear lack of implementation of the security mechanisms recommended by the standard. Moreover, with the exception of automotive embedded networks, critical embedded networks are isolated from the outside world.

In the following, we will focus on robustness to failures. Indeed, one of the obstacles to the deployment of synchronous networks in critical embedded systems is the impact of a failure of the synchronization mechanism. It is therefore necessary to prove that synchronization cannot be lost and that a failure must be mitigated within a bounded time. In the literature, we note that there is little work on the subject of robustness to failure. We explained this by the fact that BMCA dynamically mitigates failures. A few works, however, highlight the long reconfiguration times involved in BMCA due to it complexity. As far as we know, the literature has not yet examined the pros and cons of static port state configuration. Indeed, this solution was proposed very recently with the 2020 version of the standard, and certain points will only be completely standardized with the publication of P802.1ASdm. Moreover, it doesn't seem suited to the needs of sectors such as factory automation or residential and professional A/V due to the size of the network and the need for a plug-and-play solution. Whereas in a critical embedded network, the designer has full control over the network, which is relatively small compared to the factory automation network.

So to answer the need for robustness in critical embedded networks, Part III of this manuscript will start by a comparison between BMCA and static configuration to determine which of these two solutions is best suited to the critical embedded network we're studying. Then a second chapter will be dedicated to the design of such a static configuration. And as already said, the security aspect will not be addressed further in this thesis.

3.3.3 gPTP impact on other traffic

The literature highlights the very low impact of network load on synchronization precision, thanks to the use of Quality of Service (QoS) mechanisms to prioritize synchronization traffic over the rest of the so-called useful traffic. However, to the best of our knowledge, the opposite, i.e. the impact of gPTP message on useful traffic, has never been evaluated.

Part IV explores the impact of gPTP on useful traffic through two points, putting into practice the previous contributions. The first point is the evaluation of the impact of precision and gPTP traffic on the Time-Aware Shaper (TAS), more precisely the bandwidth lost due to the over-dimensioning of the TAS windows to take account of synchronization imprecision. The impact of the dimensioning parameters of precision, determined in Part II, and robustness, determined in Part II, will be studied in a TAS deployment scenario on the satellite case study. This chapter will also be the opportunity to address the need for TAS in terms of precision. The second point is the evaluation of the impact of these previously mentioned parameters on worst-case network traversal latencies. Indeed, as we saw earlier, the flows that cross a critical network must cross it in a bounded time to satisfy the needs of the applications that consume them. However, the configuration of the synchronization protocol inevitably has an impact on the quantity of gPTP messages in the network, which can disrupt the traversal of useful flows. This impact will also be studied in a practical application of the spatial case study.

Part II

Precision

Chapter 4

Simulating IEEE802.1AS accurately

In order to give answers to the achievable precision evaluation question, first investigations were conducted by simulation. Indeed, simulation allows to study very finely a protocol : you can instrument any type of event, you can experiment with many parameters or test new mechanisms that may not be available on commercial hardware implementations.

However, although simulation is a very useful tool, it is necessary to make sure that it is representative of reality. In this chapter, we start with an open source simulation library, add the missing gPTP mechanisms, and then calibrate it with commercially-available TSN hardware to ensure results accurately depict a realistic implementation. Results confirm the fidelity of the simulator for both 100Base-T and 1000Base-T physical layer, as evidenced by the RMSEs of approximately 3 ns between sliding average of the precision measured and simulated.

We have chosen to work with the library of [81]. Indeed, we have seen in Chapter 3 that many works have used simulation libraries [61], [72], [51] and [62] but only 2 are open-source ([90] and [81]). The simulation library of Wallner et al. [90] is of higher complexity than the one of Puttnies et al. [81] due to the implementation of many PTP mechanisms that are not part of IEEE802.1AS. Therefore, we decided to carry out this work on the basis of the library of Puttnies et al. [81] that supports the core functions of IEEE802.1AS by design.

Moreover, as seen in Chapter 3 with the work of Loschmidt et al. [74][73], sources of inaccuracy exist due to the implementation of the physical layer. No available simulator finely captures the physical layer impact identified by Loschmidt et al. As such, we show in this chapter the benefit in integrating these sources of inaccuracy in the library of Puttnies et al. Therefore, we challenge simulation results with extensive measurements using a 100 Base-T physical layer to validate and improve the simulation accuracy. We show that is it possible to extend this simulation model to other types of physical layer technologies such as done for a 1000Base-T physical layer. These results [39] were published at ERTS 2022.

4.1 Puttnies et al.'s simulation library

In [81], Puttnies et al. describe their work on a gPTP simulation library based on OMNeT++ 5.2 and INET 3.6.3. The authors implemented a module called EtherGptp, which is embedded in the INET protocol stack between the Ethernet encapsulation module and the mac module. This

module implements the base functionalities described in IEEE802.1AS. The library implements the message exchange mechanism for measuring the peer-to-peer propagation delay and the logic for computing the *pdelay* using the four timestamps as described in section 2.5. However, the logical syntonization isn't implemented. The distribution of `Sync` and `Follow_Up` and the offset computation is implemented according to the standard's recommendations. This distribution of synchronization messages is based on a static configuration of port state. As far as the clock is concerned, the authors have chosen to implement a simple constant drift clock model. They explain that this constant drift is not representative of reality, as drift can vary with time due to external conditions, but as this variation is slow, it remains realistic in short simulations.

The library's correct operation is justified by a comparison with the results obtained by Lim et al. in [72]. The authors point out that their library estimates the propagation delay more accurately than the Lim et al. library, measuring an error of 0.5ns between the actual delay and the estimated delay, compared with up to 10ns for Lim et al. The precision obtained with the simulation library is not discussed in the paper.

4.2 gPTP Simulation model extensions

In order to compare the simulation precision with real measurements on TSN switches, we first had to do some enhancements. These enhancements can be divided into two axes. The first is compliance with the standard. Indeed, the logical syntonization mechanism is not implemented in the Puttnies et al. library, although it is described in the standard, and is therefore present in TSN-capable devices. The implementation of this mechanism is described in sub-section 4.2.1. The second axe of enhancement is the introduction of a realistic inaccuracies model. Indeed, all the measurements and computations performed with the library are perfect, except for rounding errors, but on real devices there are many sources of inaccuracies that affect the precision of the synchronization protocol. Thus in the sub-section 4.2.2, the changes made to add realistic sources of inaccuracies are described.

4.2.1 Logical syntonization

Logical syntonization is an essential mechanism to reach higher levels of synchronization precision. Indeed, it allows to take into account the local drift compared to the Grandmaster one using the *neighborRateRatio* when updating the *correction field*. Without this mechanism, the relative drift between the Grandmaster and the time aware system is not compensated for, causing a wrong estimate of the correction to be applied to the clock. This error is also propagated to the other devices with the *correction field*.

We have added this syntonization step following the IEEE802.1AS standard to the simulator of [81]. The addition of this mechanism implies the implementation of *neighborRateRatio* and *rateRatio*. The *neighborRateRatio* computation has been implemented using the timestamps t_3 and t_4 of the *pdelay* mechanism as described in the latest version of the standard. The implementation of *rateRatio* has also led to changes in the `Follow_Up` structure and in the *correctionField* computation.

4.2.2 Towards a more realistic inaccuracies model

To better capture the real behavior of devices supporting IEEE802.1AS, the following sources of imprecision have been added to the simulator.

Clock granularity It is the duration between two increments of the clock counter. When hardware timestamping is implemented, the typical value of this granularity is of a few nanoseconds (often 8ns

or 10ns). Thus, each timestamp measured in the IEEE802.1AS protocol undergoes an error between 0 and one granularity. Since synchronization mechanisms rely on measuring delays (i.e. differences of timestamps), e.g. a duration of the propagation delay computation or the residence time of a `Sync` message, any delay undergoes an error between minus one granularity and one granularity.

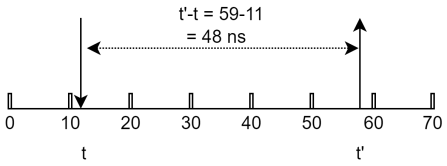


Figure 4.1: Impact of clock granularity on a duration measurement.

The naive example in Figure 4.1 pictures the delay between the reception time t and transmission time t' of a message. Without granularity, this duration is 48ns. With a granularity of 10ns, the clock reading is of 10 at reception time and of 50 at transmission time, leading to a delay of 40ns instead of 48ns.

To implement this behavior in the simulator, we round every timestamp to the immediately lower multiple of the configurable clock granularity.

Communication model As described by Loschmidt et al. in [74], the second source of inaccuracies for a synchronisation protocol is the physical layer. These inaccuracies can be separated in two sources : a jitter and a constant asymmetry. Here only the details will be given for the 100Base-T. 1000Base-T jitter and asymmetry will be briefly explained with the results.

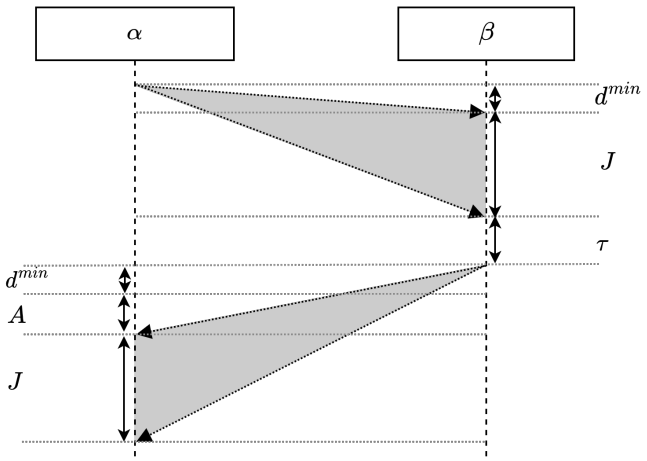


Figure 4.2: Illustration of the 100Base-T simulation communication model.

Despite the hardware timestamping used in devices supporting IEEE 802.1AS which eliminates software-related jitter, the physical layer causes an implementation dependent jitter. This jitter is the sum of the jitter which takes place between the transmission timestamps and the sending of the message on the link and the jitter which takes place between the reception of the message and the reception timestamps. For 100Base-T, Loschmidt et al. observed a total physical jitter that follows a normal law not depending on the direction and of standard deviation 0.286ns. Figure 4.2 illustrates

this physical jitter J. Indeed, when α communicates with β, all messages undergo a delay between the transmission timestamp and the reception timestamp that varies between d^{min} and $d^{min} + J$ according to a normal law.

Furthermore, a constant propagation delay asymmetry may exist. For 100Base-T physical layer, its magnitude depends on the initialization phase. This asymmetry is rooted in the Phased Lock Loop (PLL) system that may lock on dissimilar edges of the signal at the RX interface on both ends of the link. If the PLLs of the two RX interfaces at both ends of the same link lock on dissimilar edges, then the propagation delay in both ways is asymmetric due to a longer bits bufferization in one side, which causes an error in the estimation of the propagation delay. The 5 possible edges being 8ns apart, the worst asymmetry is therefore of 32ns. On Figure 4.2, this asymmetry A is undergone in the $\beta \rightarrow \alpha$ direction. Thus, a message traveling in $\beta \rightarrow \alpha$ will take Ans longer than a message traveling in the opposite direction.

These two physical layer phenomenons have significant impact on the synchronization results since they change the propagation delay statistics and values. This error on the *Pdelay* leads to errors in the estimation of the time spent by the `Sync` messages on the link and therefore inaccuracies in the synchronization.

In terms of changes in the library, we have created a new class *Channel100BaseT* which inherits from *cDatarateChannel* (the class previously used) at the level of the communication channel in order to add these notions of random jitter and edge that can cause an assymetry. These two additions are also configurable in the files describing the simulation.

Oscillator and PLL noise There are other sources of inaccuracy such as PLL or oscillator noise. Work have been devoted to study the oscillator noise as summarized by Leeson D. in [70]. This noise is composed of five type of colored noise with a configurable predominance factor. Such noises are implented in libPLN as part of LibPTP made by Wallner et al. [90].

However, with respect to our representativeness objective, such noises pose two limitations. The first one is the parameterization of these noises. Indeed, the predominance factors will vary for each oscillator while requiring highly specialized means of measurement to determine them. And secondly, the impact of these noises are very small, less than one nanosecond [74]. Thus in our case we decided to neglect them. But for very precise systems, like an atomic clock, such noise simulation are more usefull.

4.3 Experiments

In the previous section, we detailed the changes made to improve the simulation library's realism. This section aims to calibrate and validate the choices made during the implementation and to identify calibration steps to improve simulation accuracy. Therefore, we measure and analyse first, on the real TSN switch, the behavior of the switches clock. Second, we challenge the results obtained by the propagation delay measurement mechanism with the simulated ones. Finally we compare the precision of the synchronization IEEE802.1AS to the simulation results to validate the simulator's accuracy. From the first two steps, we extract calibration steps that can be performed to adjust the simulator to a specific 802.3 physical layer technology.

4.3.1 Experimental setup

The goal is to calibrate and validate the behavior of the simulation library using real devices. Configuration parameters of IEEE802.1AS are given in Table 4.1. The library is configured with the

Protocol Parameters	Value
Sync Interval	0,125s
Pdelay Interval	1s
SyncLocked	True

Table 4.1: Protocol parameters

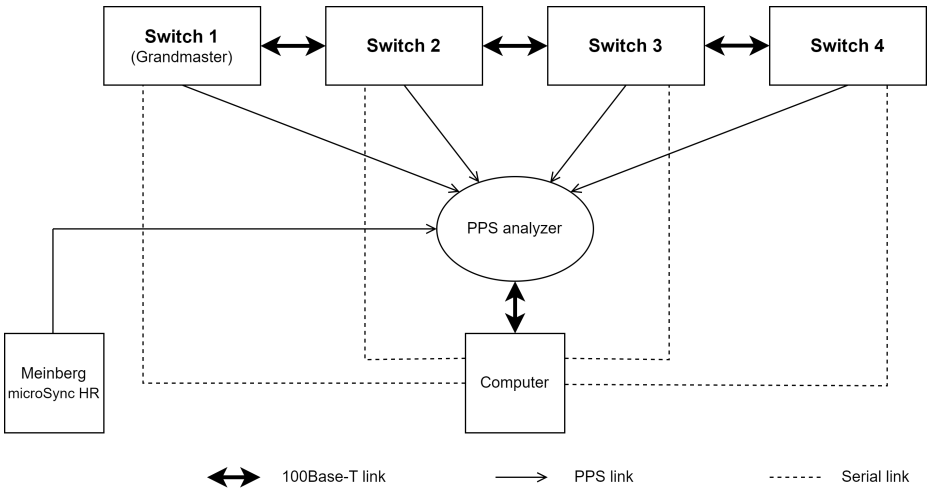


Figure 4.3: Experimental measurement topology.

help of the values determined in the rest of this chapter to get as close as possible to the behavior of real device.

Our experimental testbed, pictured in Figure 4.3 consists of:

- 4 Fraunhofer IPMS TSN Multiport Switch Core - TSN-SW v0.5.0 on Netleap boards,
- a netTimeLogic PPS Analyer,
- a Meinberg microSync HR,
- 100Base-T Ethernet link.

As described in Figure 4.3, the four switches are connected in a chain topology. We capture the progress of the different clocks using the PPS Analyzer. One of our experiments described later requiring greater accuracy needs the use of a better quality reference clock as a reference for the PPS Analyzer. Thus, we used a Meinberg microSync HR slaved on GNSS time connected to the reference input of the PPS Analyzer with a PPS link. Finally, to configure and retrieve internal switch values, such as the result of the *Pdelay* computation, we use the serial port of the switches to communicate with a Linux computer.

Simulations also consider the same topology. Unlike a one-hop topology, this multi-hop topology allows to confirm a representative computation of the reality for the *correctionField*. The measurements are carried out by OMNeT ++ at instants following precise events, e.g. the end of the propagation delay measurement or just before and just after the correction of the clock.

To determine the duration of experimentation and simulation necessary to obtain significant results, we study the evolution of the mean squared error (MSE) between the normalized distribution of values obtained by the *Pdelay* mechanism of a given experiment duration compared to an experiment duration of 32h. A duration of 3600s corresponds to a MSE of around 10^{-5} of the number

of occurrences as shown in Figure 4.4. Figure 4.5 makes it possible to observe the small difference between the distribution of value obtained by the *Pdelay* mechanism after 1 hour of experience and the distribution after 32 hours. On this example, we compute a MSE of 3.8e-06 between the two distributions. The sources of variability, such as the granularity and the physical jitter, do not depend on the AS mechanism which is studied. Therefore we will use this duration of experimentation for the other mechanisms and not only for the *Pdelay* mechanism. Thus, for the rest of this study, we take measurements during 3600s.

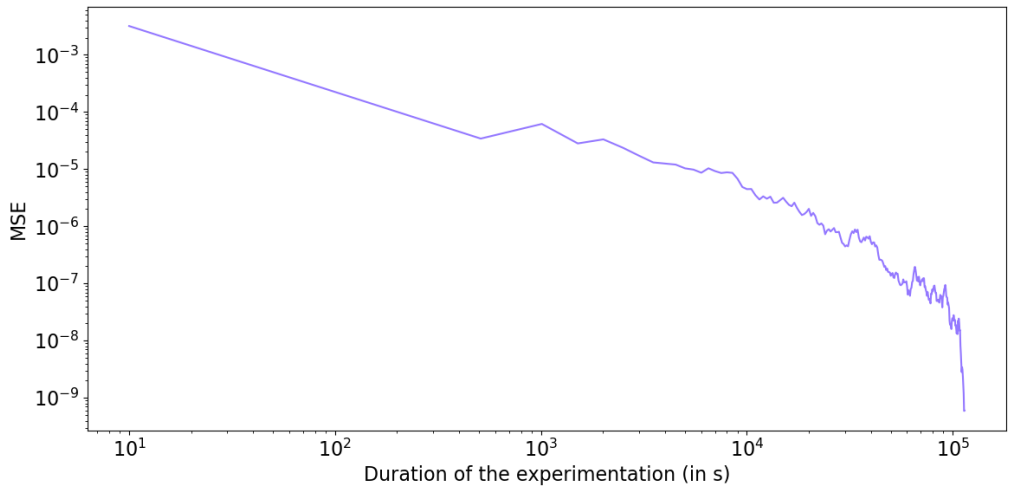


Figure 4.4: MSE of the distribution of the results of the *Pdelay* mechanism according to the duration of the experience compared to a 32 hours experience with the real switches

The following sections present the three experiments that we carried out to compare the operation of the simulator with the operation of real devices in order to calibrate and validate this simulation model.

4.3.2 Clock calibration

In [81], Puttnies et al. have chosen to implement a simple constant drift clock. In order to validate this choice and adjust the parameters of the simulator clock deviation in relation to our real devices, we have studied the behavior of the clocks of our switches in freerunning during one hour. To do this, we measured with the help of the PPS Analyzer the evolution of the drift for each switch compared to the Meinberg microSync HR slaved on GNSS when the synchronization is deactivated. We use the Meinberg microSync HR on this experiment for its clock quality.

Figure 4.6 and Table 4.2 show the results of this experiment. These measurements were taken 10 minutes after starting the devices, so that the temperature of the electronic components is stable and without attempting to control the room temperature. As indicated in Figure 4.7, presenting multiple results of the linear regressions carried out, similar results are obtained for the different repetitions of the measurements during 24h.

The r-values presented in the Table4.2 are very close to 1 and show a very strong correlation between the 3600 measurement points and a linear function. The p-values, very close to 0, confirm this observation, while affirming that the number of measurement points is sufficient to conclude on

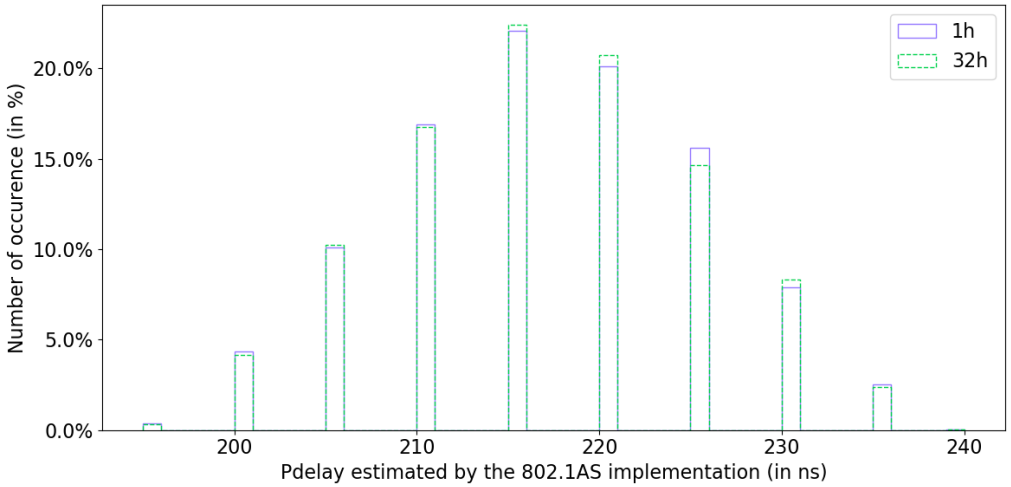


Figure 4.5: *Pdelay* distribution for an 1-hour experiment and a 32-hour experiment

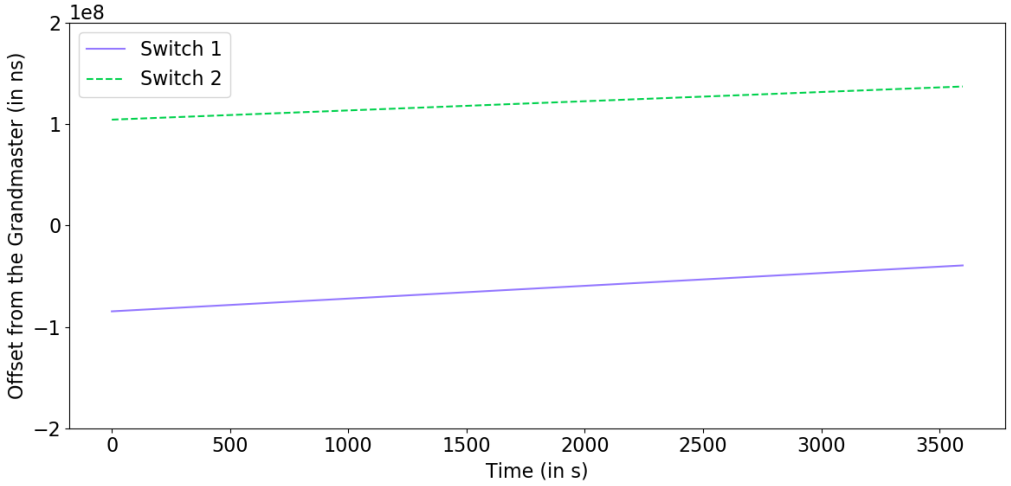


Figure 4.6: Grandmaster clock offset of two switches during 3600s

	Switch 1	Switch 2
ppm	12.593	9.094
p-value	$< 10^{-9}$	$< 10^{-9}$
r-value	0.99999995	0.99999999

Table 4.2: Results of linear regression of the data presented in Figure 4.6

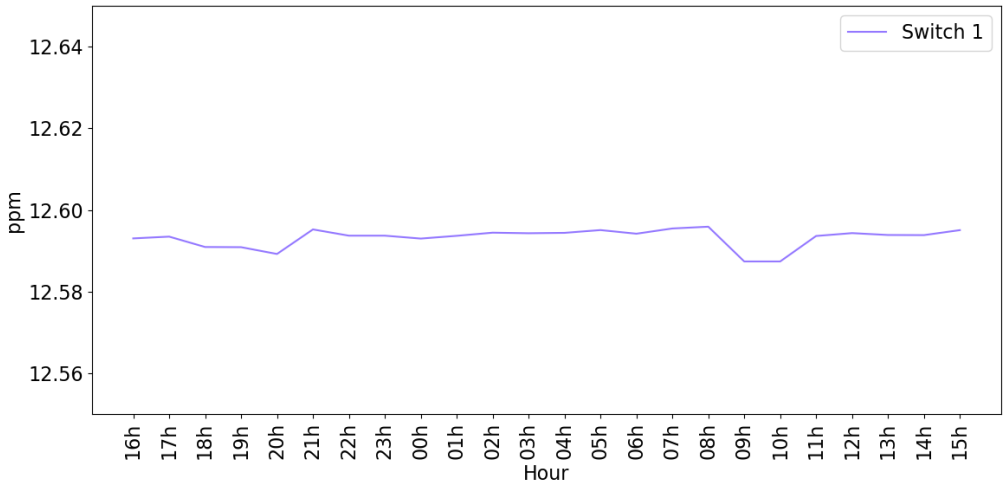


Figure 4.7: Result of linear regression of switch 1 clock drift measured for 3600s and repeated for 24 hours.

the linear behavior of the evolution of the drift of our clocks during measurement. Futhermore, the slope resulting from this regression is the drift of our clocks in freerunning. It is thus possible to state that the constant drift clock implementation in the simulator is representative of our switch clocks over 1 hour in an environment where temperature variations are small. The linear function resulting from the linear regression are shown in Figure 4.6.

However, by repeating the experiment during 24h, we observe small variations in ppm within a range of 0.009 ppm , as shown in Figure 4.7 caused by the temperature variation in the room. In order to use up to date ppm value in the simulator, in the rest of our comparison between simulation and reality, we perform a ppm measurement in freeruning compared to the Grandmaster of the experiment before performing any other 1-hour measurements on real devices.

4.3.3 Canal calibration

In order to adjust and validate the different sources of inaccuracy that we have added to the simulator, we measure the distribution of the values obtained by the *Pdelay* mechanism without any filter and compare them to the value obtained with the simulator. To configure the simulator, we use the relative drift measured before the experiment and the mean link delay measured during the experiment.

In order to determine the standard deviation to parameterize the normal distribution which adds jitter to the link crossing time in the simulator, we compare the distributions obtained using the simulated and real *Pdelay* mechanism. Figure 4.8 shows the MSE obtained between the simulated and measured distribution of the *Pdelay* as a function of the standard deviation used to parameterize the normal law which causes the physical jitter when crossing the link. By minizing the MSE, we find a standard deviation of 12.5ns, which is quite different from the 0.286ns measured by Loschmidt et al in [74]. Indeed, our switch embeds a different physical layer chip from the one used for their measurements. In addition, our measurement takes place much further in the chain of message transmission. Indeed, our measurement is based on timestamps which take place between the MAC

layer and the physical layer of the OSI model while their measurement takes place directly at the physical level using the RX_DV and TX_EN PINs of the MII. Thus other sources of jitter can be found between their point of measurement and the clock (where we do our measurement). In our case, the objective is to finely simulate clock-to-clock jitter of our platform and not the MAC to MAC jitter as measured by Loschmidt, so we'll use our results in the following.

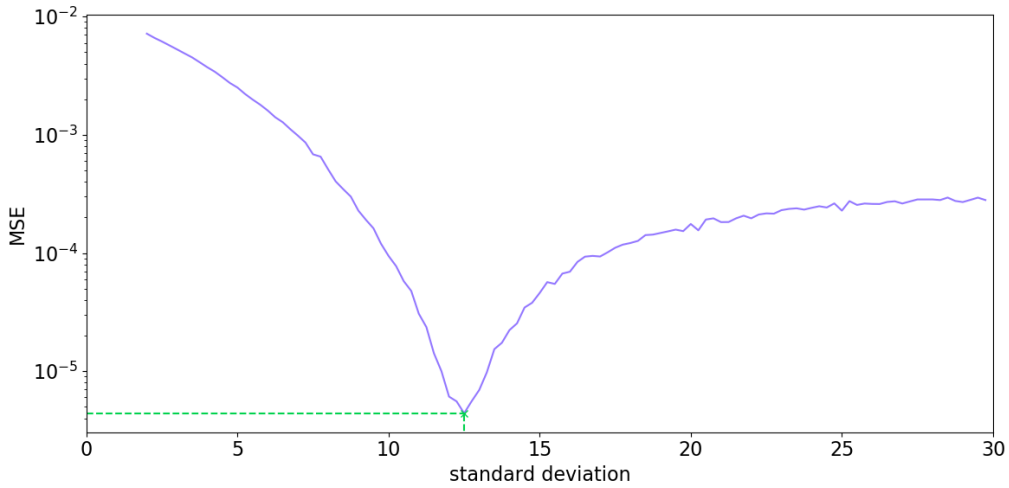


Figure 4.8: MSE between the simulated and measured distribution of the *Pdelay* during 32h according to the standard deviation used to simulate the physical jitter in the simulator

Figure 4.9 presents the results, after canal calibration, obtained when switches are synchronized and exchange *Pdelay* messages every seconds. It shows the distribution of the different values obtained by the measurement mechanisms of the link delay of one of the switches and of the simulation library obtained during 32h. For this measurement, we use a measurement duration much longer than what we have previously determined to get a distribution that captures as much rare events as possible. The simulation parameters such as mean link delay, granularity and physical standard deviation were chosen to match the experimental distribution. By analyzing the distribution of the propagation delays obtained by the real device, in Figure 4.9, we can deduce the granularity of the clocks used by observing the difference between the two consecutive values. Here, the difference being 5ns, the granularity is therefore 10ns, because of the division by two in the *Pdelay* formula (2.8). As can be seen with this figure, our simulation gives a distribution very close to reality although a little more pessimistic than reality but which does not bother us in view of our critical on-board scope of application.

When the link between two measurements is reinitialized, a variation of the mean *Pdelay*, as shown in Figure 4.10, is observed. These two distributions originate from a periodic measurement of the *Pdelay* on real devices for which we trigger a link reinitialization every hour. As described by Loschmidt et al. in [73], the difference between the two distributions is a consequence of link delay asymmetry. This asymmetry is induced by the edges on which the PLLs of each physical interface lock during link initialization. If the two PLLs don't lock on the same edge, the various messages of the *Pdelay* mechanism undergo a different link propagation delay depending on their direction on the link. As such, there is an error in the estimation of the link delay by the *Pdelay* mechanism since it is based on a symmetrical channel assumption. With the library, we reproduce this behavior by

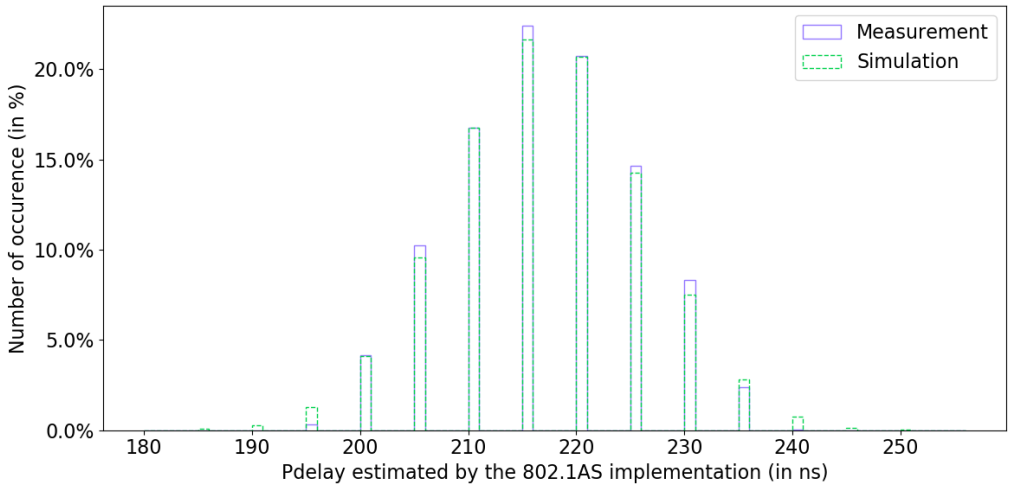


Figure 4.9: Distribution of the results obtained by the simulated and real *Pdelay* mechanisms

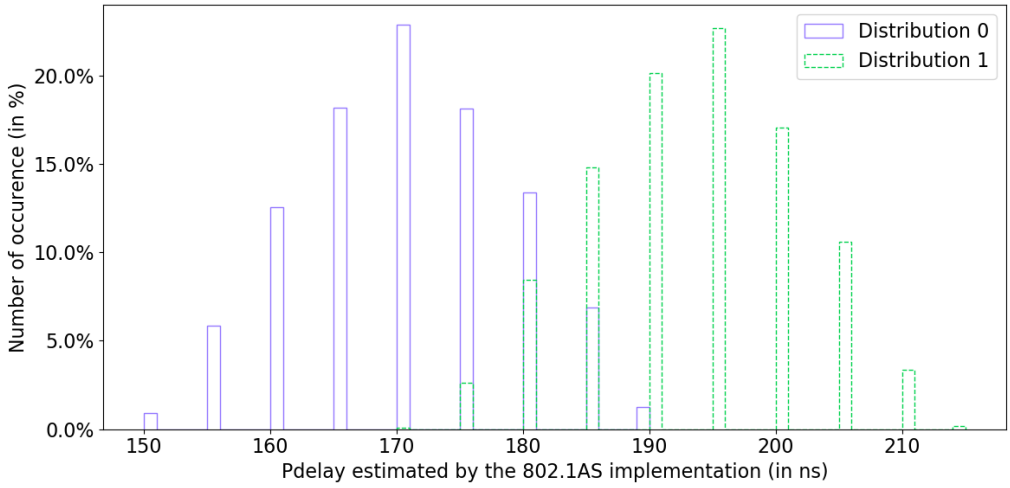


Figure 4.10: Distribution of the *Pdelay* values obtained during two experiments with the real switches.

randomly setting the edge on which each interface is locked for each simulation.

4.3.4 Validation

In this last part, we compare the synchronisation precision measured with the TSN switches to the one computed by the OMNeT ++ simulator. This steps validates the different enhancements of the simulation library and the calibration steps proposed in this chapter.

Bounding the real precision

The measurements are triggered by the PPS Analyser every second. The `Sync` messages are sent about every 125ms. The PPS measurements aren't synchronized with the `Sync` dates and as such, the measurements may be taken at various times of the synchronization cycle. For instance, if the previous synchronization stage takes place a few nanoseconds before the measurement time, the precision measured using the PPS signal is much better than if the last synchronization stage takes place a few tens of milliseconds before.

Unlike real measurements based on the PPS signal, we can measure in simulation the synchronization precision at specific times which are related to the main protocol execution steps. The worst precision can thus be measured just before the synchronization procedure takes place and the best one just after it. Since we can't compare measurements and simulation at exactly the same dates, we plot the best and the worst synchronization by simulation, and the real measurements in the same figure to validate whether measurements lie in between best and worst simulation precision.

Figure 4.11 shows the precision measured on the first hop, as well as the simulated precision before and after clock correction for 3600s. With this figure, we observe that the simulator allows to bound the result obtained with the real device on the first hop, despite the limitations of the PPS signal. These results are reasonable, even though they don't allow us to judge the accuracy of the bounds obtained with the simulator.

Accuracy of simulated precision bounds

By repeating the previous experiment multiple times, we have sometimes observed gaps in the precision measured using the PPS signal as shown in the figures 4.12, 4.13 and 4.14. These figures show the measured precision, as well as the simulated precision before and after synchronization respectively at the first, second and third hops. These PPS gaps represent the situation where we move from measuring precision just after synchronization takes place to measuring precision in reality just before synchronization takes place. Over a campaign of 200 one-hour measurements, we have observes these PPS gaps 19 times.

To determine the cause of these gaps, we studied the *preciseOriginTimestamp* of the `Sync` emitted by the Grandmaster using a traffic capture. We observed that the average time between the `Sync` is not 125ms but 125.000541ms. These extra 541ns cause a shift of the sending instant of the `Sync` compared to the cycle of measurement of PPS. With such an average time shift between two `Syncs`, it takes 8h01 for a time shift of 125ms to occur and to allow the observation of the passage of the correction of the clock before the emission of the rising edge of the PPS to after this edge.

Anyways, close to the PPS gap, it's possible to determine when the synchronization cycle takes place. In this situation, it becomes possible to compare the bounds obtained with the simulator with the worst and best precision measured around the PPS gap. With the figures 4.12, 4.13 and 4.14, it can be seen that the simulator makes it possible to precisely bound the synchronization precision reachable with these TSN switches. Indeed, we observe the best case, just after the synchronization, and the worst case, just before the synchronization, in a single measurement for

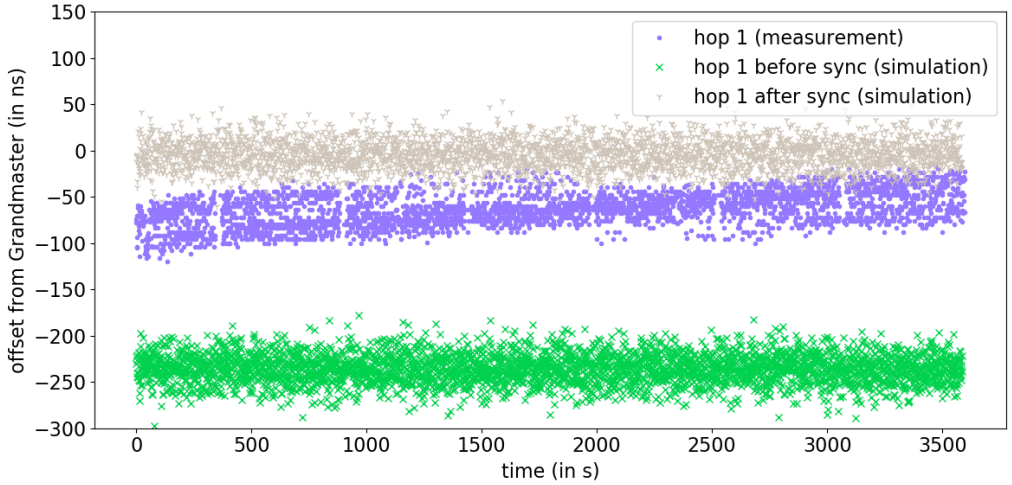


Figure 4.11: Offset between the Grandmaster clock and the switch clock at hop 1 in reality and the simulator. The simulator is configured with the granularity, mean link delay and the standard deviation estimated in the previous experiments. For the clock drift, we use the drift measured before each experiment.

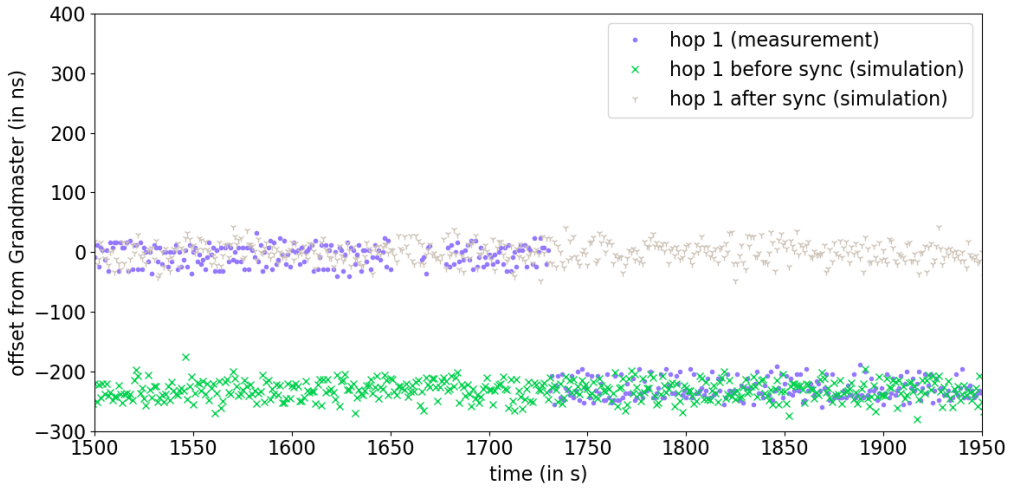


Figure 4.12: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop 1 in reality. Simulated best and worst offset are plotted as well.

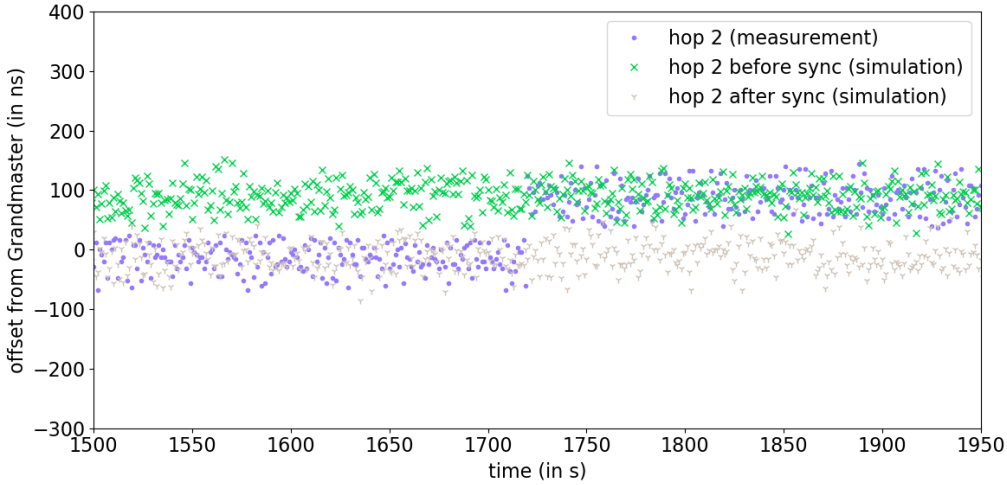


Figure 4.13: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop 2 in reality. Simulated best and worst offset are plotted as well.

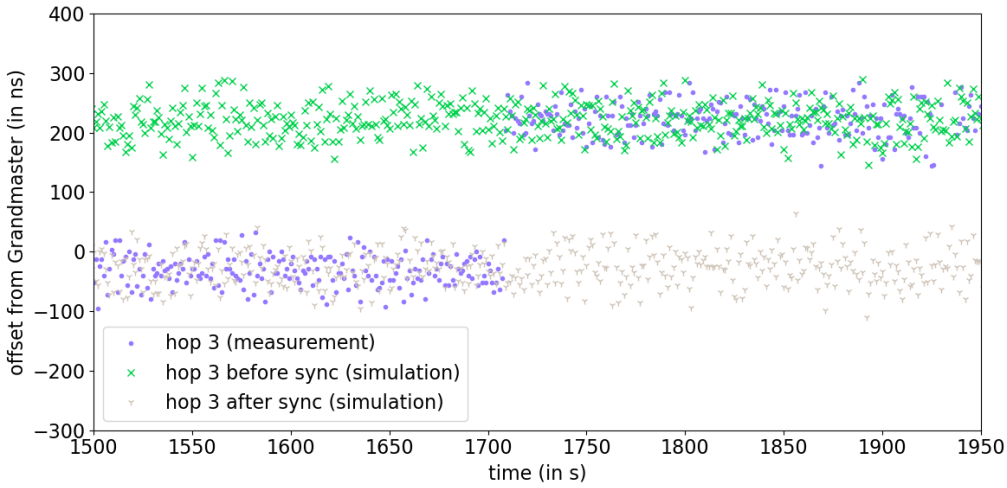


Figure 4.14: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop 3 in reality. Simulated best and worst offset are plotted as well.

	Hop 1	Hop 2	Hop 3
Minimal RMSE (ns)	0.89	1.1	2.1
Maximal RMSE (ns)	7.1	4.8	6.2
Median RMSE (ns)	2.3	2.8	3.9
Mean RMSE (ns)	3.4	2.9	4.2

Table 4.3: Result of the RMSE calculation using 10 different simulations

	Switch 2	Switch 3	Switch 4
ppm	-1.850	0.817	1.997

Table 4.4: Clock drift of the different devices measured before the simulator validation experiment

the three hops. Furthermore, the measurements and comparisons with the simulation for hop 2 and 3 make it possible to validate the implementation of the calculation of the *correctionField* and thus the measurement of the `Sync` residence time. We also observe that the dispersion of the precision values remains similar despite our use of a pessimistic physical jitter.

These observations are validated by the computation of the RMSE between the measured and simulated sliding average of the synchronization precision. Sliding average is computed over a window of 150 samples for the 500 samples preceding the PPS gap for each one of the three hops. Due to the progressive shift of the synchronization cycle relatively to the PPS measurement time, we are constrained to use a small window. This small window isn't large enough to capture all possible variations in simulation. To compensate for this variability, we perform an RMSE calculation over 10 different simulations. Results are are presented in the Table4.3. As shown in this table, the average RMSE is approximately 3 ns. There is also an increase in the median RMSE as the number of hops increases. This increase in the differences between reality and simulation is mainly caused by the pessimistic estimation of the physical jitter that is accumulated at each hop, introduced during the calibration phase of the *Pdelay* measurement mechanism.

Using the three figures, we also observe a large variation in the precision before synchronization as a function of the number of hops. This variation is due to the relative drift between the Grandmaster and the device in question. Indeed, before synchronization, the precision error is mainly caused by the drift which has taken effect since the last synchronization, here since 125ms. Before this measurement, we measured the relative drift between the Grandmaster and the different devices and observed a lower drift for switch 3 (hop 2) than for the other devices as shown in the Table4.4.

4.4 Extension to 1000Base-T

4.4.1 Implementation specific inaccuracies

On the 1000Base-T side, Loschmidt in [73] observed a jitter that is different depending on the direction which they explain by the master/slave architecture of this physical layer. Indeed, unlike 100Base-T which uses two twisted pairs per direction, 1000Base-T uses the 4 twisted pairs available on a cat-5 twisted-pair wire for both directions. To allow full duplex, this physical layer depends on a mechanism called echo cancellation. Such a mechanism relies on a synchronization of the physical interfaces on both sides of the same link. This synchronization relies on a master sending its physical signal (containing encoded data or idle symbols) on which the slave interface synchronizes by recovering the clock from the data line. A message sent by the master physical interface then undergoes a jitter following a normal law as for the 100Base-T. A message sent by the physical slave

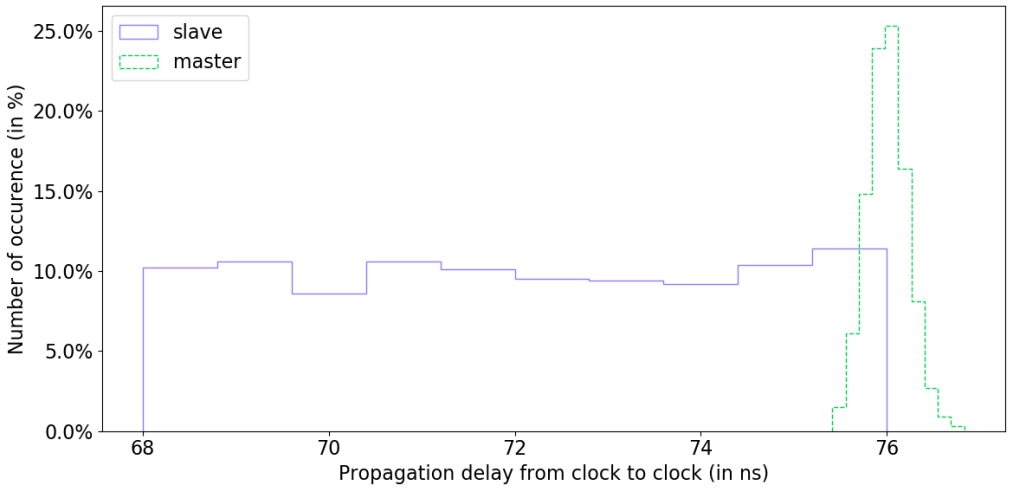


Figure 4.15: Distribution of the propagation delay of messages crossing a 1000Base-T link according to the characterization of Loschmidt et al. and its physical state

interface undergoes a uniform jitter between 0 and 8ns which is explained by the difference between two edges on the signal used by the slave to synchronize.

An asymmetry is naturally caused by the fact that these two jitters are not centered on the same average delay as shown in Figure 4.15. On this figure, we observe the distribution of the delays undergone by the messages according to their interface of arrival after implementation on the simulator and using the standard deviation measured by Loschmidt et al i.e. 0.216ns.

To implement this physical layer on the simulator we have created a new channel *Channel1000BaseT* which, as for the 100Base-T, inherits from *cDatarateChannel*. The two new jitters are also configurable in the files describing the simulation.

4.4.2 Results

Before comparing the precision obtained with the simulator with the precision measured with the switches during the use of the 1000Base-T, it is necessary to carry out the two stages of calibration as with the 100Base-T.

The first calibration step concerns the drift of the clocks. As we use the same switches as before, the drift of the clocks can also be approximated by a linear drift for the duration of our experiment i.e. 1 hour.

The second calibration phase aims at characterizing the physical jitter and the granularity of the clocks. As for the characterization of the drift, the granularity is identical no matter the physical layer. For the physical jitter, the objective is to determine the standard deviation of the master normal law. To do this, we have applied the same method as for 100Base-T which consists in minimizing the MSE between the real and simulated distribution of the values of *Pdelay* by varying the standard deviation used by the simulation. The evolution of the MSE is presented in the Figure 4.16. The superposition of the two 1-hour distributions of *Pdelay* after calibration of the simulator is presented in figure 4.17.

The minimum MSE obtained is $3e{-}4$ns for a standard deviation of 4.95ns. Compared to 100Base-

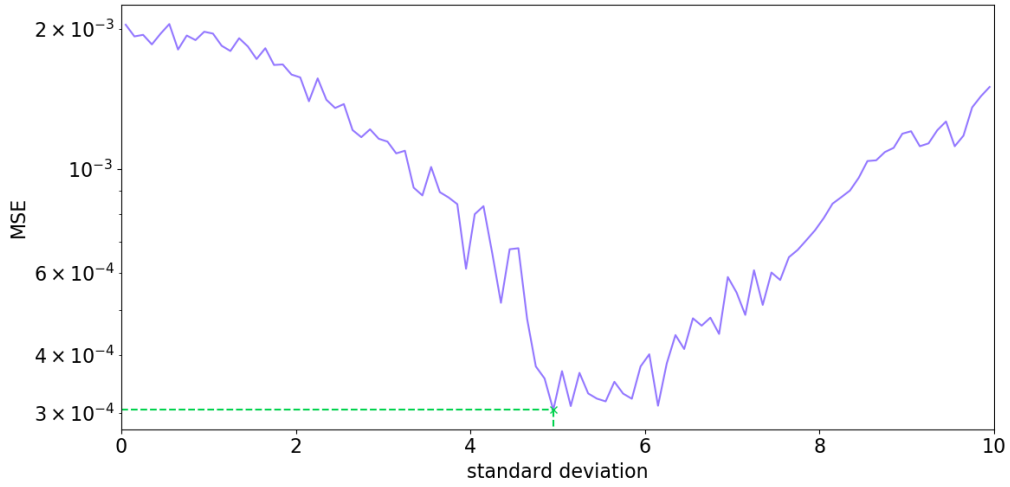


Figure 4.16: MSE between the simulated and measured distribution of the *Pdelay* with a 1000Base-T physical layer during 11h according to the standard deviation used to simulate the physical jitter in the simulator

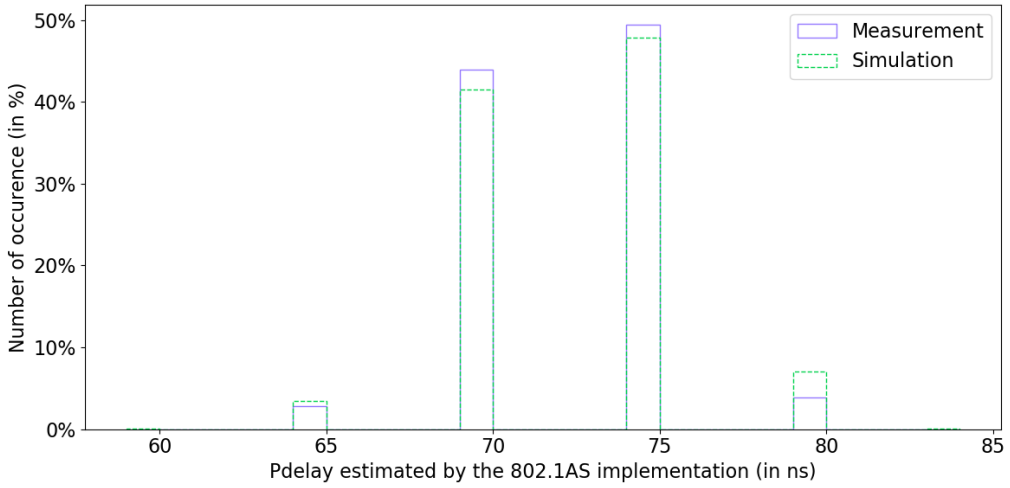


Figure 4.17: Distribution of the results obtained by the simulated and real *Pdelay* mechanisms with 1000Base-T physical layer

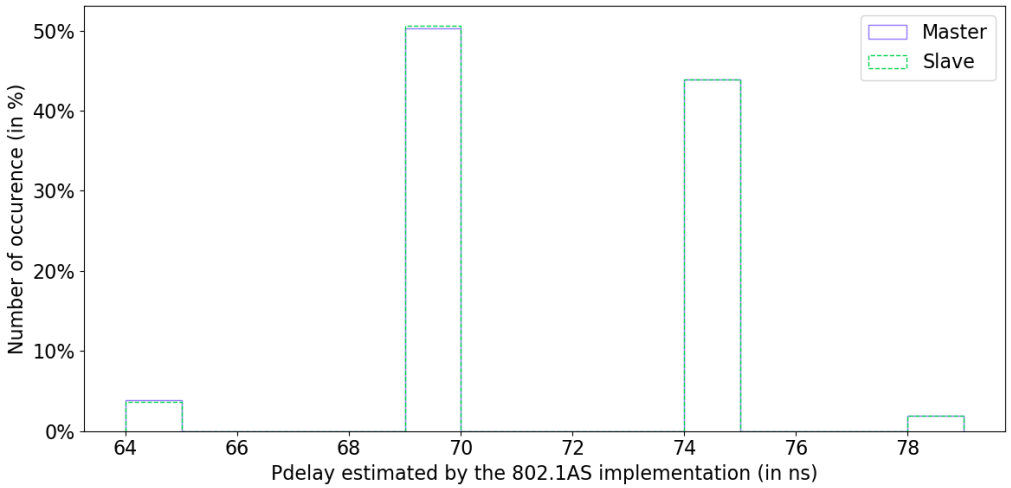


Figure 4.18: Distribution of the *Pdelay* obtained by the simulation for master and slave 1000Base-T physical layer state

T, we observe a greater variability of the MSE as a function of the standard deviation, which can be explained by the smaller amount of Pdelay measured on the hardware (32h vs 11h). When this standard deviation is used in the simulator, as in Figure 4.17, we observe that the distributions are visually very similar, which is confirmed by an MSE of $7e-5$ between the two distributions. However, as for the 100Base-T, the simulation is slightly more pessimistic with minimum and maximum 5ns larger than reality.

As for the master or slave status of the physical interface, once the simulator was calibrated we varied it to study its impact on the *Pdelay*. Figure 4.18 presents the result of this experiment. We can see that there is no significant impact on the distribution of *Pdelay*. We will therefore not give it any importance in the following.

Now that the simulator is calibrated to be representative of the 1000Base-T physical layer, we can study the precision in terms of representativity. To do so, we performed 90 1-hour experiments with a single hop due to hardware availability constraints. During these 90 experiments we observed 17 PPS gaps. Using the simulator, we reproduced six of these 17 experiments. One of them is depicted in the Figure 4.19. On this figure, we observe a good representativity of the simulation before and after the PPS gap. This representativeness is confirmed by an RMSE, calculated as for the 100Base-T, of 3.2ns for the experiment represented in the figure and an average RMSE over the 6 reproduced experiments of 3.9ns. Moreover, compared to 100Base-T, we observe that the dispersion is lower when using 1000Base-T thanks to the lower jitter of this physical layer for the material we use.

Conclusion

This section presents the enhancement of an open source simulation library of the IEEE802.1AS synchronization protocol. The integration of logical syntonization and real hardware inaccuracies brings the simulation library closer to reality. Our tests show the fidelity of the simulator for both 100Base-T and 1000Base-T physical layer after calibration compared to the result obtained with

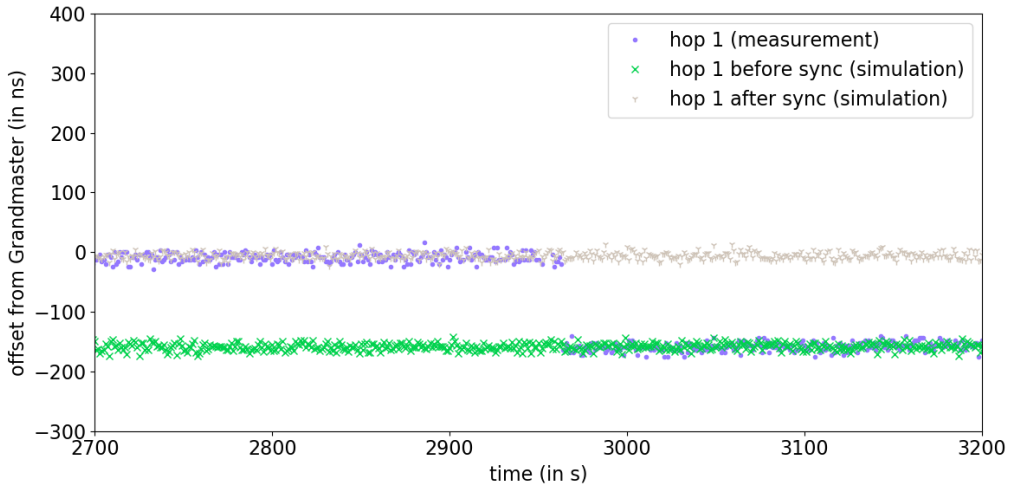


Figure 4.19: Identification of a PPS gap in the offset measurement between the Grandmaster clock and the switch clock at hop 1 when using a 1000Base-T physical layer. Simulated best and worst offset are plotted as well.

real TSN switches, as evidenced by the MSEs in the order of $7 * 10^{-5}$ of occurrence between the distribution of values obtained by the simulated and measured link delay mechanisms, as well as the RMSEs of approximately 3 ns between the sliding average of the measured and simulated precision. This library now allows to study the precision of the synchronization and its impact on the clients according to parameters such as topology, protocol configuration, filters for measuring propagation delay and clock servo algorithm. Our changes have been released in open-source and are available at the following link : `https://github.com/irit-rmess/gPtp-implementation`. In addition, we also propose a method which is repeatable to calibrate the simulator for other switches or end stations and other physical layer.

However, the main limitation of this work and a track of improvement for the future is the study of the probability law of jitter. Indeed, although we have achieved a sufficient representativity for our needs by using a single probality law on both sides of the link, it is very likely that in reality it is a combination of several laws. We assume that there would be at least a combination of two laws: one between the physical layers of a same link (the one studied by Loschmidt et al.) and one between the physical layers and the clocks. The possibility to instrument a TSN device in order to study precisely the different jitter would be an interesting plus to reach an even more excellent representativeness of the simulation.

Chapter 5

Bounding the worst-case precision

We have seen in the previous chapter that we can evaluate the performance of the synchronization via a simulator calibrated for the hardware used. Although interesting to understand and configure our network, the simulation is far from being a proof of correct synchronization precision when we talk about a critical network. For the latter, validation and sometimes certification steps are necessary. These steps are generally based on formal methods to evaluate the system in the worst-case. For our systems colleagues, these proofs are brought by worst-case execution time (WCET) analysis. In the world of critical networks, proofs on worst-case latencies/jitters are usually obtained from network calculus analysis, as done for the AFDX certification.

For a use of gPTP in such a context, a proof of worst-case precision is needed for the dimensioning of distributed applications but also for network mechanisms relying on a common clock like the Time-Aware Shaper. As mentioned in Chapter 3, a publication [62] has been devoted to this subject by Gutiérrez et al. using 100Base-T that lacks a validation proof. This is a valuable proposition that we propose to extend and refine in order to support more types of physical layer, but also to reduce the pessimism of the bound on the worst-case precision. Comparisons with the realistic simulator of the previous chapter, experimental measurements and exhaustive worst-case research show a good representativeness of the hardware and a very reduced pessimism compared to the state-of-the-art model [62], but also compared to the observed worst-case with at best an overestimated bound of only 5.4%. These results [41] were presented at ETFA2023.

5.1 Modelling sources of inaccuracies

In this section, we introduce a generic system model that we leverage to carry out the formal development of the worst-case precision bound of Section 5.2. The point of our model is to capture different sources of inaccuracies leading to synchronisation imprecisions. Thus, we model the timing behaviours of the network and the time-aware systems related to *i*) the physical inaccuracy of clocks like drift and granularity and *ii*) the communication delay variability induced by the physical layer implementation of the network interface card as seen in the previous chapter.

5.1.1 Clock model

Clock drift ρ

Oscillators are imperfect in the sense that their oscillation frequency does not stay constant over time. This frequency variation, also called drift rate, is generally measured in *parts per million (ppm)* defined by the number of seconds the local clock deviates in a million seconds of the reference time. The accuracy of an oscillator is characterized by a bound on this drift rate. For instance, an oscillator characterized with +10ppm (respectively -10ppm), may run up to 10μs faster (respectively slower) with respect to a perfect time every second. Practically, the drift varies over time due to aging or external conditions, such as temperature.

For the worst-case calculation, we assume that the clock undergoes a constant drift rate given by its oscillator upper bound (10ppm for instance). A clock can therefore be modelled with Equation (5.1) where t_i is the time on the time-aware device i, t_p the perfect timescale, ρ_i the bound on the drift rate of the oscillator of i and I the interval since the last synchronisation.

$$t_i = t_p + \rho_i \times I \tag{5.1}$$

This drift can be mitigated by periodic re-synchronization using IEEE802.1AS for instance. However, re-synchronization is prone to multiple inaccuracies that we model in the following.

Clock granularity G

The other essential feature inducing timing errors in clocks is the granularity G. As explained in chapter 4, it is the duration between two increments of the clock counter. Thus, each timestamp measured in the IEEE802.1AS protocol undergoes an error between 0 and G. Since synchronization mechanisms rely on measuring delays (i.e. differences of timestamps), any delay undergoes an error between $-G$ and G.

5.1.2 Communication model

IEEE802.1AS works through message exchanges over a wired transmission channel (in our case) whose time of transmission and reception have to be measured as precisely as possible. But as we have seen in the previous chapter, besides depending on the granularity, the precision of this timestamp also depends on the physical layer technology. Implementation-specific features of the physical layer technology impact the accuracy of transmission delay measurement, as highlighted by the experimental work of Loschmidt [73]. Although the propagation delay on the link is constant, the delay between the message timestamping and its actual transmission or between the reception and its timestamping varies due to the hardware implementation and transmission technology. As described in Chapter 4, it triggers two kinds of inaccuracies:

- A physical jitter J that varies over time according to a given distribution. On the same link, Loschmidt's measurements show that the distribution of this jitter may depend on the direction of the communication. For instance, for 1000Base-T, the delay follows a uniform distribution on one direction and a normal one on the other direction, with different widths each. For 100Base-T, the jitter follows in both directions the same normal distribution.
- A constant link asymmetry latency A that induces an always larger delay in one communication direction compared to the other one. This constant latency can be related to various technological choices. For instance, a link layer using an optical fiber where a different wavelength is used per direction induces an asymmetric propagation delay. The link initialisation of a 100Base-T is another reason of constant asymmetry, as observed by Loschmidt in [73].

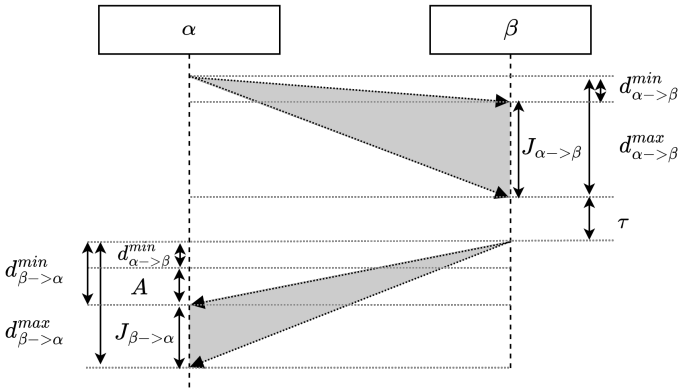


Figure 5.1: Illustration of the communication model.

In order to capture this refined characterization of the communication delay, we propose to extend the communication model used in Chapter 4 for the simulation to support worst-case analysis on any physical layer. The main features of this new communication model are depicted in Figure 5.1. It is characterized by a directional communication delay and jitter, a link asymmetry latency and a residence time. Numerical values have to be set according to the physical layer characteristics.

Directional communication delay and jitter We assume an asymmetric communication delay. Thus, the delay is defined for a communication direction. For two time-aware systems α and β, the delay $d_{\alpha\to\beta}$ for a communication direction $\alpha \to \beta$ belongs to an interval given by the smallest and largest possible delays: $d_{\alpha\to\beta} \in \left[d^{\min}_{\alpha\to\beta}, d^{\max}_{\alpha\to\beta}\right]$. The size of this interval is defined as the jitter $J_{\alpha\to\beta}$:

$$J_{\alpha\to\beta} = d^{\max}_{\alpha\to\beta} - d^{\min}_{\alpha\to\beta} \tag{5.2}$$

This jitter can be set according to a safe characterization of its width distribution for a target PHY layer from extensive measurements similar to the ones done in chapter 4. Note that for the physical jitter following a normal law, we consider in the following that it is bounded between $-3\times\sigma$ and $3\times\sigma$, with σ the standard deviation of this law. Such a decision allows to cover 99.7% of the possible values of the distribution. However, due to our pessimistic physical jitter standard deviation estimation in the previous chapter, we never observe an overrun of this bound despite our extensive measurement batches. For an even safer approach, it is possible to use a larger multiplier for σ to compute J, in order to increase the coverage of possible values. For example, the coverage of $J = 6 \times \sigma - (-6 \times \sigma)$ is 99.999998%. But such an overly safe approach does not allow for a fair comparison with reality, as we do in the following.

Similarly, the delay $d_{\beta\to\alpha}$ and corresponding jitter $J_{\beta\to\alpha}$ are defined for the direction $\beta \to \alpha$. Directional communication delays and jitters are illustrated on Figure 5.1 for the case where $J_{\alpha\to\beta}$ is larger than $J_{\beta\to\alpha}$

Link asymmetry latency A In the case of an asymmetrical propagation channel, a constant latency A can be added to the delay of one of the directions to account for the additional propagation delay experienced. In Figure 5.1, a link asymmetry latency $A > 0$ is added for direction $\beta \to \alpha$. Thus:

$$d^{\min}_{\beta\to\alpha} = d^{\min}_{\alpha\to\beta} + A \text{ and } d^{\max}_{\beta\to\alpha} = d^{\min}_{\alpha\to\beta} + A + J_{\beta\to\alpha} \tag{5.3}$$

Residence time τ Any back-to-back request-response synchronization mechanism, such as the one used for the `Pdelay` computation, necessitates some processing time on the responder side before transmission. This processing time is typically called the residence time and denoted τ.

5.2 Bounding the worst-case precision

This section gives the main developments leading to the computation of safe upper and lower bounds.

5.2.1 Upper and lower bounds on synchronization precision

The instantaneous synchronization error $P_i(t)$ of a time-aware system i is the difference between its estimate $t_i(t)$ of the Grandmaster clock and the Grandmaster clock $t_{GM}(t)$:

$$P_i(t) = t_i(t) - t_{GM}(t) \tag{5.4}$$

Our goal is to derive a safe upper and lower bound on the synchronization precision of a time-aware system i. Let P_i^U (respectively P_i^L) be the upper (respectively lower) bound on precision of the system i. These bounds are constructed to guaranty the following constraints:

$$P_i^U \geq \max_t P_i(t) \quad \text{and} \quad P_i^L \leq \min_t P_i(t) \tag{5.5}$$

As depicted in Figure 5.2, this precision depends on two quantities:

- the relative clock drift between the Grandmaster and time-aware system i since the last synchronisation point (in green on the figure),
- the wrong estimation of the Grandmaster time by the time-aware system i at the previous synchronisation point, which is due to the implementation-specific sources of inaccuracy modelled before (in purple in the figure).

Let's note $E_{drift_i}(t)$ the clock drift at time t between the Grandmaster and the time-aware system i since the last synchronisation point. This drift can be bounded between $-E^U_{drift_i}$ and $E^U_{drift_i}$. Let's also denote δGM_i the error in the estimation of the Grandmaster clock at the last synchronisation point. As shown later, calculating upper and lower bounds on δGM_i differs slightly. As such, we differentiate notation δGM_i^U and δGM_i^L accordingly. Consequently, upper and lower bounds on synchronization precision are given by:

$$P_i^U = E^U_{drift_i} + \delta GM_i^U \tag{5.6}$$

$$P_i^L = -E^U_{drift_i} + \delta GM_i^L \tag{5.7}$$

The rest of Section 5.2 gives the main developments leading to our final results, and more specifically the derivation of $E^U_{drift_i}$ and ${\delta GM_i}^U$.

5.2.2 Derivation of $E^U_{drift_i}$

The worst drift occurs when the clocks of the Grandmaster and the time-aware system i drift in opposite directions, e.g. the Grandmaster clock deviates on the positive side while the time-aware system i deviates on the negative one. This drift is eliminated at each synchronisation point, and its magnitude increases until the next synchronisation point. Therefore, the largest possible drift is

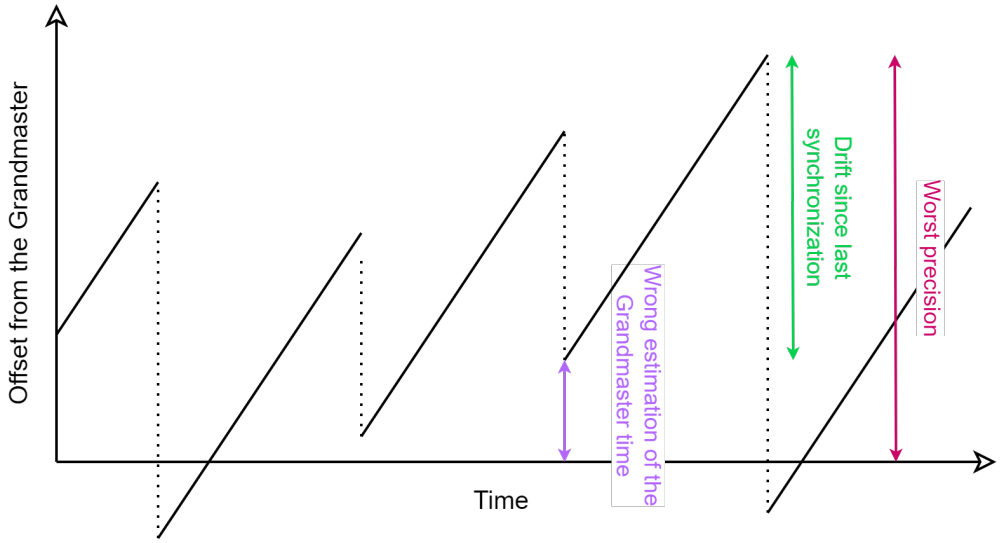


Figure 5.2: Illustration of the two quantities that impact the offset between the Grandmaster and a time-aware system as a function of time.

observed right before a synchronisation point. Let's assume that previous synchronization occurred I time units ago. The largest possible drift $E^U_{drift_i}$ is given by:

$$E^U_{drift_i} = I \times (|\rho_i| + |\rho_{GM}|) \tag{5.8}$$

As expressed in Equation (5.8), this drift is proportional to the delay I since last synchronization occurred. In an ideal situation, I would be equal to *syncInterval* I_s. However, in practice, `Sync` and `Follow_Up` can be delayed by other messages in the switch queues, possibly at each hop. Thus, I_s which is technically the delay between two consecutive `Follow_Up` receptions, is an optimistic value. The worst situation is when the first `Follow_Up` message undergoes the smallest possible network traversal delay, while the second one undergoes the largest possible one. In this case, the delay I is the sum of the *syncInterval* I_s and the largest network jitter J_{fup} that the `Follow_Up` message can experience : $I = I_s + J_{fup}$.

If the topology of the network, the port queuing disciplines (TAS, CBS, etc.) and flows (periodicity, traffic class, etc.) are known, J_{fup} can be upper bounded using a network calculus analysis as done by Zhao et al. in [91].

5.2.3 Derivation of δGM_i^U

δGM_i^U (respectively δGM_i^L) represents an upper bound (respectively lower bound) on the Grandmaster's time estimation error on the time-aware system i made at the last synchronization point. Next, we develop the construction of δGM_i^U in details, and provide a more concise description of its lower bound counterpart.

Let's consider the time when the time-aware system i receives a `Follow_Up` message. Let's denote this time t^{fup}_{GM} if expressed in the Grandmaster reference clock and t^{fup}_i if expressed in the clock of time-aware system i. Upon `Follow_Up` message reception, the time-aware system i is able to

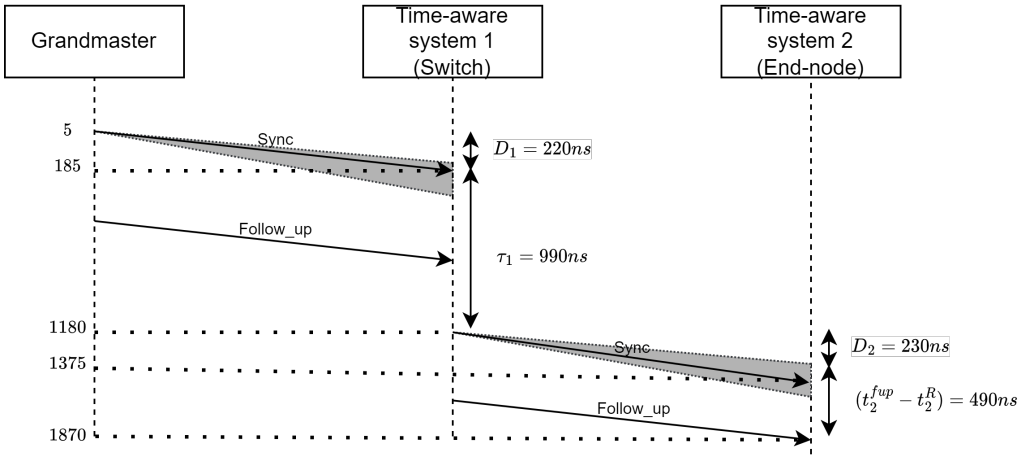


Figure 5.3: Naive illustration of errors that impact GM_i. Clock granularity for all systems is 10ns. Each value is given in the time base of its time-aware system.

calculate its estimation of the Grandmaster current time using Equation(2.14) recalled below.

$$GM_i = O + C_{i-1} + D_i + (t_i^{fup} - t_i^R) \tag{5.9}$$

By definition, δGM_i^U is upper-bounding the error between its estimation of the Grandmaster clock and t_{GM}^{fup}:

$$\delta GM_i^U \geq \delta GM_i \quad \text{with} \quad \delta GM_i = GM_i - t_{GM}^{fup} \tag{5.10}$$

This error is illustrated in Figure 5.3 where the synchronization initiated by the transmission of a `Sync` message sent by the Grandmaster finishes when the end-node receives the last `Follow-Up`. Note that the values D_1, τ_1, D_2 and $(t_2^{fup} - t_2^R)$ of Figure 5.3 are given in the local time base of the corresponding time-aware system. We also recalled that D_i is the *pdelay* between time-aware systems i and $i-1$, τ_i the `Sync` residence time in time-aware system i and $(t_i^{fup} - t_i^R)$ the duration between `Sync` reception and `Follow_Up` on the time-aware system i. In the time base of the Grandmaster, the `Sync` is transmitted at time 5ns and the last `Follow_Up` message is received at time 1870ns since the communication delay is of 180ns for the first hop and 195ns for the second hop, and the residence time in the switch is of 995ns and the delay $t_2^{fup} - t_2^R$ is of 495ns.

In the same figure, we notice that D_1, the estimation of the delay by the *Pdelay* mechanism is erroneous for time-aware system 1: 220ns instead of 180ns. Similarly, time-aware system 2 over-estimates D_2: 230ns instead of 195ns. Moreover, errors induced by the clock granularity trigger an over-estimation of GM_i. Indeed, the end-node calculates a value of GM_i equal to 1930ns, leading to an error of $\delta GM_i = 60ns$. The clock granularity of the switch induces an under-estimation of τ_1 and the clock granularity of the end-node an under-estimation of $t_2^{fup} - t_2^R$. Moreover, even though the initial `Sync` is transmitted at 5ns, its respective `Follow_Up` carries a value $O = 0$ns because of the clock granularity of the Grandmaster.

The worst synchronization error is observed when the synchronization protocol triggers an estimate of the `Sync` traversal time that is as large as possible compared to the smallest possible `Sync` network traversal delay. Formally, we can express the bound on the synchronization error as the sum of bounds on the errors induced by the different components of GM_i:

$$\delta GM_i^U = \delta O^U + \delta C_{i-1}^U + \delta D_i^U + \delta(t_i^{fup} - t_i^R)^U \tag{5.11}$$

with δD_i^U the upper bound on the *Pdelay* error and δC_{i-1}^U the upper bound on the *correctionField* error. Both errors originate from the *Pdelay* mechanism. The *correctionField* error originates as well from errors on the *rateRatio* and on the *residence time* estimation. Bounds on δO^U and $\delta(t - t_i^R)^U$ are a consequence of the granularity on timestamps readings.

In the model of [62], δO is neglected and $\delta(t - t_i^R)$, the duration between `Sync` reception and instant of the clock correction on the time-aware system i, is not accounted for. In our version, $\delta(t - t_i^R)$ captures the more precise two-step mode of IEEE802.1AS. Our derivation of δD_i^U differs from [62] since it captures the communication channel asymmetries, the *neighborRateRatio* error and finer residence time error. The derivation of δC_{i-1}^U follows the one of [62] but its numerical values change since it relies on δD_i^U.

Bounding the *Pdelay* error with δD_i^U

We are looking for δD_i^U, the largest value of δD_i, estimated for the time aware system i using a *Pdelay* message exchange with its time parent node j. We have $\delta D_i^U = D_{worst} - D_{best}$, with D_{worst} the worst estimation of the *Pdelay* and D_{best} the best case, which happens when the result of (2.8) matches the real minimum link communication delay. Thus, we have:

$$D_{best} = d_{j\to i}^{\min} \tag{5.12}$$

As a reminder, the *pdelay* and *neighborRateRation* equations are recalled below.

$$D = \frac{(t_2 - t_3) + nr(t_4 - t_1)}{2} \tag{5.13}$$

$$nr = \frac{f_{req}}{f_{resp}} = \frac{t'_3 - t_3}{t'_4 - t_4} \tag{5.14}$$

To compute δD_i^U, it boils down to maximizing D_{worst}. Since D_{worst} follows (5.13), we have to derive the conditions where nr and $(t_4 - t_1)$ are maximal and $(t_3 - t_2)$ is minimal. As such, we derive first the highest possible estimation of nr, then the maximum estimate of $(t_4 - t_1)$ and the minimum estimate of $(t_3 - t_2)$.

Study of nr Our objective is to find the highest possible overestimation of *neighborRateRatio* δnr_i^U. Thus, we define δnr_i^U such as $\delta nr_i^U = nr_{i_{worst}} - nr_i$, with $nr_{i_{worst}}$ the worst possible value of the *neighborRateRatio* and nr_i the error-free one. Computing $nr_{i_{worst}}$ is equivalent to maximizing the numerator $t'_3 - t_3$ and minimizing the denominator $t'_4 - t_4$, compared to the real delay between the events that led to these timestamps.

$t'_3 - t_3$ being the difference between two timestamps internal to a time-aware system, the maximum error between $t'_3 - t_3$ and the real duration which elapsed between the two consecutive transmissions of `Pdelay_resp` can only be of a granularity unit as explained in section 5.1.1. As such, the maximum error of $t'_3 - t_3$ is $+G$.

As for the minimum error between $t'_4 - t_4$ and the real duration between the reception of the two consecutive `Pdelay_resp`, in addition to depending on the granularity G, this error also depends on the physical jitter. Indeed, as illustrated in Fig, 5.4 - Left, the variation of the propagation delay due to physical jitter can lead to the smallest value of $-J_{j\to i}$. In practice, the worst $t'_4 - t_4$ delay happens if the first `Pdelay_resp` message experiences a propagation delay of $d_{j\to i}^{\max}$ and the timestamp t_4 is taken exactly on a clock tick, while the second one experiences the smallest propagation delay $d_{j\to i}^{\min}$ and t'_4 is taken an arbitrary small instant before a clock tick.

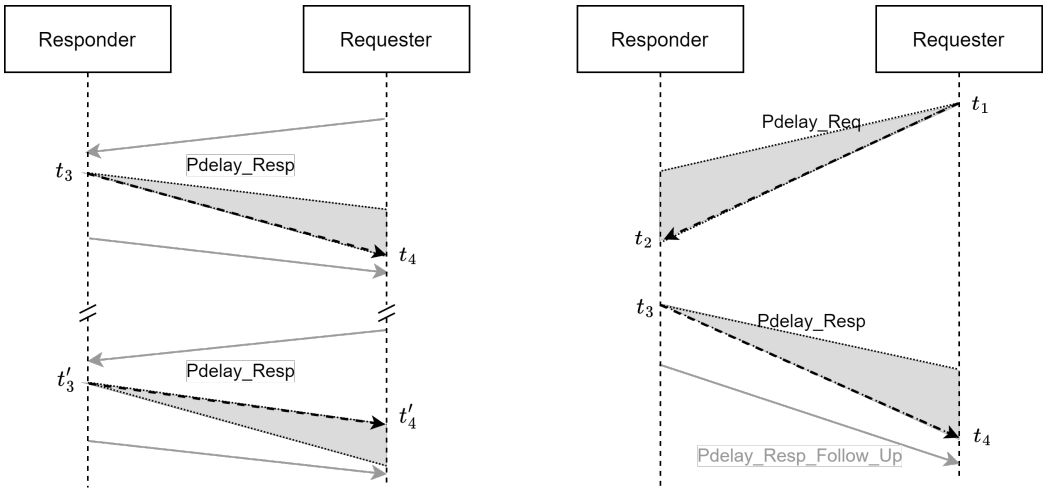


Figure 5.4: Left: Illustration of the `Pdelay_resp` propagation delay variation with jitter J (solid gray) and the worst-case propagation delay scenario for the *neighborRateRatio* computation (black dashed arrow)
Right: Illustration of the `Pdelay_req` and `Pdelay_resp` propagation delay variation due to jitter J (solid gray) and the worst-case propagation delay scenario for the *Pdelay* computation (black dashed arrow)

Finally, clock drift also has an impact on the computation. A maximum positive drift ρ_j in the responder increases $t'_3 - t_3$ because the clock is faster than reality. Conversely, a maximum negative drift $-\rho_i$ decreases $t'_4 - t_4$. Overall δnr is :

$$\begin{aligned}\delta nr_i^U &= nr_{i_{worst}} - nr_i = \frac{t'_3 - t_3 + G}{t'_4 - t_4 - G + (d_{j\to i}^{\min} - d_{j\to i}^{\max})} - \frac{t'_3 - t_3}{t'_4 - t_4} \\ &= \frac{I_p + I_p \times \rho_j + G}{I_p - I_p \times \rho_i - G - J_{j\to i}} - \frac{I_p + I_p \times \rho_j}{I_p + I_p \times (-\rho_i)} = \frac{2G + G(\rho_j - \rho_i) + J_{j\to i}(1 + \rho_j)}{I_p(1 - 2\rho_i + \rho_i^2) + (\rho_i - 1)(G + J_{j\to i})}\end{aligned} \tag{5.15}$$

Study of $t_4 - t_1$ We aim at maximizing $t_4 - t_1$ which is the duration between the transmission of a request and the reception of a response message. Similarly to the $t'_4 - t_4$ case in the nr computation, $t_4 - t_1$ is impacted by the granularity G, the variable propagation delay and the physical asymmetry A. Adding the granularity G maximizes $t_4 - t_1$ (like for $t'_3 - t_3$). The maximum propagation delay $d^{\max}$ is sudden twice: once for the `Pdelay_req`, once for the `Pdelay_resp`, as illustrated in Figure 5.4-Right. Finally, the impact of the physical asymmetry A is experienced on the direction $i \to j$, towards GM.

Study of $t_3 - t_2$ Finally, to minimize $t_3 - t_2$ which is a duration between two internal events of a time-aware system, we have to remove the granularity G to the real duration.

Calculus of δD_i^U To summarize, the bound δD_i^U is:

$$\begin{aligned}\delta D_i^U &= \frac{\max(t_4 - t_1)(nr_i + \delta nr_i^U) - \min(t_3 - t_2)}{2} - d_{j\to i}^{\min} \\ &= \frac{[(\tau_i + 2d_{j\to i}^{\min} + J_{j\to i} + J_{i\to j} + A)(\rho_i + 1) + G](nr_i + \delta nr_i^U) - [\tau_i(1-\rho_j) - G]}{2} - d_{j\to i}^{\min}\end{aligned} \tag{5.16}$$

Bounding the *correctionField* error

The *correctionField* C_{i-1} in a time-aware system $i-1$ is computed by Equation (2.13) recalled below.

$$C_i = C_{i-1} + D_i \times r_{i-1} + (t_i^S - t_i^R) \times r_{i-1} \times nr_i \tag{5.17}$$

It depends on the *correctionField* and the *rateRatio* in the previous system $i-2$, the *neighborRateRatio* and the *Pdelay* between systems $i-2$ and $i-1$ and the *residence time* in $i-1$. Bounds on the worst-case values of *neighborRateRatio*, *Pdelay* and *residence time* (with $t_3 - t_2$) have been set in part 5.2.3.

The *rateRatio* is computed by $r_i = r_{i-1} * nr_i$. For the rest of the paper, we assume (as done in [62]) that all time-aware systems are the same. Thus, they have the same clock with the same drift rate bounds, granularity and the same physical interface with the same physical jitter and asymmetries. So we have $nr_1 = nr_2 = ... = nr$ and $\delta {nr_1}^U = \delta {nr_2}^U = ... = \delta nr^U$. This assumption allows the following simplification: $r_i = nr^i$. To calculate the bound on *rateRatio* overestimation $\delta {r_i}^U$, the same method as before is applied with the following derivation:

$$\delta r_i^U = (nr + \delta nr^U)^i - nr^i = i \times nr^{i-1}\delta nr^U + ... + i \times nr(\delta nr^U)^{i-1} + (\delta nr^U)^i \tag{5.18}$$

In order to simplify this equation, the powers of δnr are neglected since they are very small compared to the main term $i * nr^{i-1}$ as done in [62] and thus:

$$\delta r_i^U \approx i \times nr^{i-1} \times \delta nr^U \tag{5.19}$$

With the parameters we use in the results Section 5.3, we get $\delta r_9^U = 0.86 \times 10^{-6}$. In this example, the powers of δnr that were neglected previously are equal to 3.2×10^{-13}.

We can now compute the correction field error δC in a time-aware system $i-1$. From Equation (5.17), we have:

$$\begin{aligned}\delta C_{i-1}^U &= C_{i-1_{worst}} - C_{i-1} \\ &= [C_{i-2} + \delta C_{i-2}^U + (d_{(i-2)\to(i-1)}^{\min} + \delta D_{i-1}^U)(r_{i-2} + \delta r_{i-2}^U) + (\tau_{i-1} + G)(r_{i-1} + \delta r_{i-1}^U)] \\ &\qquad - [C_{i-2} + d_{(i-2)\to(i-1)}^{\min} r_{i-2} + \tau_{i-1} r_{i-1}] \\ &= \delta C_{i-2}^U + r_{i-2}\delta D_{i-1}^U + \left(d_{(i-2)\to(i-1)}^{\min} + \delta D_{i-1}^U\right)\delta r_{i-2}^U + r_{i-1}G + (\tau_{i-1} + G)\,\delta r_{i-1}^U\end{aligned} \tag{5.20}$$

As for the *neighborRateRatio*, we assume that our time-aware systems use the same hardware. Thus we have $d_{GM\to 1}^{\min} = d_{1\to 2}^{\min} = ... = d_{(i-2)\to(i-1)}^{\min}$, $\delta D_0^U = \delta D_1^U = ... = \delta D_{i-1}^U$ and $\tau_0 = \tau_1 = ... = \tau_{i-1}$. In addition to this assumption, using $r_i = nr^i$ and Equation (5.19), previous Equation (5.20) simplifies to:

$$\begin{aligned}\delta C_{i-1}^U &= \delta D_{i-1}^U \left(\frac{nr^{i-2} - 1}{nr - 1}\right) + G\left(\frac{nr^{i-1} - 1}{nr - 1} - 1\right) \\ &\quad + \delta nr^U \left((d_{(i-2)\to(i-1)}^{\min} + \delta D_{i-1}^U)\sum_{j=0}^{i-2} j \times nr^{j-1} + (\tau_{i-1} + G)\sum_{j=1}^{i-1} j \times nr^{j-1}\right)\end{aligned} \tag{5.21}$$

using the following derivation :

$$\begin{aligned}
\delta C_{i-1} &= \delta C^U_{i-2} + r_{i-2}\delta D^U_{i-1} + \left(d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}\right)\delta r^U_{i-2} + r_{i-1}G + (\tau_{i-1}+G)\,\delta r^U_{i-1} \\
&= \delta C_{i-3} + (r_{i-3}+r_{i-2}) \times \delta D^U_{i-1} + (d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}) \times (\delta r^U_{i-3} + \delta r^U_{i-2}) \\
&\quad + (r_{i-2}+r_{i-1}) \times G + (\tau+G) \times (\delta r^U_{i-2} + \delta r^U_{i-1}) \\
&= (r_0 + ... + r_{i-2}) \times \delta D^U_{i-1} + (d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}) \times (\delta r^U_0 + ... + \delta r^U_{i-2}) \\
&\quad + (r_1 + ... + r_{i-1}) \times G + (\tau+G) \times (\delta r^U_1 + ... + \delta r^U_{i-1}) \\
&= \delta D^U_{i-1} \sum_{j=0}^{i-2} r_j + (d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}) \sum_{j=0}^{i-2} \delta r^U_j + G \sum_{j=1}^{i-1} r_j + (\tau+G) \sum_{j=1}^{i-1} \delta r^U_j \\
&= \delta D^U_{i-1} \sum_{j=0}^{i-2} nr^j + (d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}) \sum_{j=0}^{i-2} \delta nr^U \times j \times nr^{j-1} + G \sum_{j=1}^{i-1} nr^j + (\tau+G) \sum_{j=1}^{i-1} \delta nr^U \times j \times nr^{j-1} \\
&= \delta D^U_{i-1} \left(\frac{nr^{i-2}-1}{nr-1}\right) + G\left(\frac{nr^{i-1}-1}{nr-1} - 1\right) \\
&\quad + \delta nr^U (d^{\min}_{(i-2)\to(i-1)} + \delta D^U_{i-1}) \sum_{j=0}^{i-2} j \times nr^{j-1} + \delta nr^U (\tau_{i-1}+G) \sum_{j=1}^{i-1} j \times nr^{j-1}
\end{aligned} \tag{5.22}$$

Bounding δO and $\delta(t_i^{fup} - t_i^R)$

Sending the `Sync` message to the i^{th} time aware system generates errors that are caused by clock granularity on two different timestamping actions.

Bound on δO The *preciseOriginTimestamp* O is a timestamp. Due to the granularity of the Grandmaster clock, it gets the value of the clock at the last tick no later than the current instant. Thus, O can be under-approximated by up to the granularity G. This erroneous timestamp is carried in the `Sync` message, but can only reduce the value of GM_i. Therefore, the largest possible over-approximation for the upper bound δO^U is of 0. However, note that for the lower bound computation, δO contributes to the worst-case situation and we have then $\delta O^L = -G$

Bound on $\delta(t_i^{fup} - t_i^R)$ The quantity $(t_i^{fup} - t_i^R)$ is the time that has elapsed between the reception time t_i^R of the `Sync` and time t_i^{fup}. It can experience an error $\delta(t_i^{fup} - t_i^R)$, because the correction does not occur at the `Sync` reception time, but later. With the two-step mode of 802.1AS, this is the case, since it is necessary to wait for the reception of the `Follow_Up` message to obtain the missing synchronization information. This error, being a duration between two internal events in the time-aware system, is thus upper bounded by the granularity G.

5.2.4 Upper bound on precision P_i^U

Finally, we can express the upper precision bound P_i^U as the sum of the drift between the Grandmaster's clock and the time-aware system's clock since the last synchronisation and errors that occurred

P_i^L	$-(\lvert\rho_i\rvert + \lvert\rho_{GM}\rvert)(I_s + J_{fup}) + \delta GM_i^L$
δGM_i^L	$\delta C_{i-1}^L + \delta D_i^L - 2G$
δC_{i-1}^L	$\delta D_{i-1}^L(\frac{nr^{i-2}-1}{nr-1}) - G(\frac{nr^{i-1}-1}{nr-1} - 1) + \delta nr^L((d_{(i-2)\to(i-1)}^{\max} + \delta D_{i-1}^L)\sum_{j=0}^{i-2} j \times nr^{j-1} + (\tau_{i-1} - G)\sum_{j=1}^{i-1} j \times nr^{j-1})$
δD_i^L	$\frac{[(\tau_i+2d_{i\to j}^{min}+A)(1-\rho_i)-G](nr+\delta nr^L)-(\tau_i(1+\rho_j)+G)}{2} - (d_{i\to j}^{\min} + J_{j\to i} + A)$
δnr^L	$\frac{-[2G+J_{j\to i}(1-\rho_j)+G(\rho_i-\rho_j)]}{I_p(1+2\rho_i+\rho_i^2)+(\rho_i+1)(G+J_{j\to i})}$

Table 5.1: Lower bound formulas on synchronization precision bound.

by estimating the Grandmaster's time on the time-aware system i:

$$\begin{aligned} P_i^U &= (\lvert\rho_i\rvert + \lvert\rho_{GM}\rvert)(I_s + J_{fup}) + \delta GM_i^U \\ &= (\lvert\rho_i\rvert + \lvert\rho_{GM}\rvert)(I_s + J_{fup}) + \delta C_{i-1}^U + \delta D_i^U + G \end{aligned} \tag{5.23}$$

5.2.5 Lower bound on precision P_i^L

A similar analysis can be performed to determine the lower bound on the worst-case precision P_i^L that can be reached when the drift of the time-aware system is negative. Opposite of δGM_i^U, the conditions leading to δGM_i^L are therefore an underestimation of the network traversal delay of the `Sync` compared to its real traversal delay. In other words, we have to minimize the error committed on the *neighborRateRatio* δnr, the *Pdelay* δD compared to the actual `Sync` propagation delay, the *rateRatio* δr_{i-1} and the *correctionField* δC_{i-1}^U to underestimate the `Sync` network traversal time. Furthermore, we also need to maximize δO, the duration between the reception of the `Sync` and the clock correction $\delta(t - t_i^R)$ and the asymmetry have to be experienced in the Grandmaster→Slave direction to maximise the actual `Sync` traversal time. Final equations for the lower bound on synchronization precision are given in Table 5.1. Note that the lower bound derivation was not addressed in the state-of-the art model of Gutiérrez et al. [62].

5.3 Results

This section starts by validating the quality of our worst-case bound by comparing it to the state-of-the-art model of Gutiérrez et al. [62] for 100Base-T technology using extensive simulations and measurements on a network representative of a typical embedded environment i.e. up to 10 hops as described in [72] for automotive, [46] for satellite, [64] for airplane networks and in our case studies. We show as well how our model scales for larger networks. Second, we study the influence of the various sources of errors on the bound in order to better engineer a tight synchronisation. Then, we instantiate the model with 1000Base-T links and compare it to the bound obtained with 100Base-T links.

5.3.1 Bound tightness validation

Simulations, exhaustive search and measurements are leveraged to address two questions: how close our bound on worst-case precision is and how it compares to the previous model of Gutiérrez et al. [62]. We show that our bound is two-times closer to the simulated worst precision observed than the

	G	ρ_{GM}	ρ_{Slave}	d^{min}	$J_{j\to i}$	$J_{i\to j}$	A	τ	J_{fup}
100Base-T	10ns	0.02ppm	10ppm	200ns	75ns	75ns	32ns	1ms	2ms
1000Base-T	10ns	0.02ppm	10ppm	200ns	29.7ns	8ns	6.85ns	1ms	2ms

Table 5.2: 100Base-T and 1000Base-T parameters.

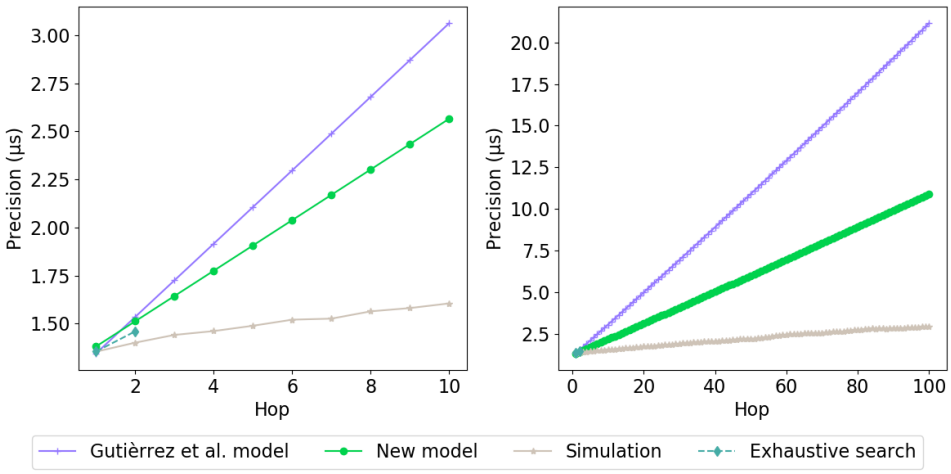


Figure 5.5: Comparison of simulated, exhaustive search and precision bound for 100Base-T physical layer according to the number of hops.

model of [62], and for a 2-hop network, follows closely exhaustive search of the worst-case. Moreover, empirical evaluation shows that our upper and lower bounds neatly frame maximum and minimum measured precision on a 3-hop network. We show as well that our bound is particularly tight for the first hops, where the model of Gutiérrez et al.[62] is optimistic.

In order to provide a fair comparison, we instantiate our model for 100Base-T links. Unless mentioned, we use the values of granularity, clock drift, propagation delay, jitter and asymmetry determined using simulation in Chapter 4. These values are specific to the switches that we use as well and thus allows a very fine comparison between the experimental measurements and the bound. Since 100Base-T jitter does not depend on the direction, we denote $J = J_{i\to j} = J_{j\to i}$. For the residence time τ, we use a common value of the literature [62]. J_{fup} is set to 2ms based on network calculus analysis using a commercially available tool[1] on our embedded automotive use-case. Numerical values are given in Table 5.2. For the protocol configuration, we use the default parameters of AS: $I_s = 0.125s$ and $I_p = 1s$. During comparison, the two models are configured with identical values.

Simulation and exhaustive search study

Figure 5.5 compares, as a function of the number of hops, our upper bound and the one of Gutiérrez et al. [62] to simulations realised with our open-source and calibrated simulation library described in 4 and to an exhaustive search as detailed later. Results are produced with the parameters of Table 5.2, except for the Grandmaster drift which is set to 0ppm (perfect clock assumption) and for

[1] https://www.realtimeatwork.com/rtaw-pegase/

J_{fup}, set to 0 as well, since we don't simulate data traffic. Simulating data traffic only makes the worst case less likely to observe, increases the search space of the exhaustive search, and is hardly precisely reproducible in the experimental test-bed.

To offer a fair comparison of both bounds, we have not taken into account the simplification of the *Pdelay* bound δD_i^U in the δGM_i^U made in the derivation of [62]. Indeed, authors neglect δD_i^U compared to the correction field error δC_{i-1}^U in δGM_i^U because they evaluate their bound on a 100-hop network. In our case, with a 10-hop network, δD_i^U is not negligible anymore. As such, we introduce δD_i^U in the model of [62] to be fair.

The simulation results presented in this comparison are obtained from a set of 400 1-hour and 10 12-hour simulations for the left part and a set of 10 1-hour and 1 12-hour simulations for the right part of Figure 5.5, for which we have randomized initial settings (initial clock desynchronization, AS mechanism start time, physical asymmetry) except for the slave clock drift which is set to the worst value (i.e. 10ppm) for fair comparison with the bounds. We have kept in the plots of Figure 5.5 the worst precision recorded at each hop among all runs.

The exhaustive search is carried out by testing all the possible combinations of the parameter values in order to determine the time of transmission or reception of synchronization messages, and deduce the timestamps and synchronization calculations. The worst offset between the Grandmaster's clock and the clock of a time-aware system is recorded across all executions. Parameters range and sampling interval are chosen as follows. For the propagation delay, the start time of synchronization, the delay between the reception of a `pdelay_req` and the transmission of the `pdelay_resp` or the delay between the transmission of a `Sync` and its corresponding `Follow_Up`, an interval of one granularity is set since it is enough to capture the worst error. The physical jitter is evaluated over its entire interval $[0, J]$. The sampling size has been chosen to get a tractable resolution and meaningful results for a 2-hop network topology. A sampling step of 1.5ns (respectively 0.05ns) for the 2-hop (respectively 1-hop) network triggers 15 billion (respectively 4.1billion) combinations.

From Figure 5.5 - Left, we observe that our bound is two times closer to the worst precision observed during the simulation. Our improvements have reduced the pessimism of our bound compared to the one of the Gutiérrez et al. Their larger pessimism is due to an overestimation of the error impacting some delays: the error related to the physical jitter is accounted for any duration while this error never happens in reality for the `Sync` residence time duration or for the duration between the reception of `Pdelay_Req` and the transmission of `Pdelay_Resp`. Moreover, the errors caused by the granularity on a duration measurement are also overestimated in the model of [62] compared to our model. This pessimism is even more obvious with a 100-hop network, as shown in Figure 5.5 - Right. After 100 hops, the state-of-the-art model reaches 21.174µs while our bound is 12.843µs. The simulation reaches 2.952 µs, which is far from the bound because the sequence of events which leads to the worst-case is less likely as the number of hops increases. The evolution of the bounds according to the number of hops being linear, in the following we focus on smaller networks more representative of embedded networks. From a complexity point of view, both models are implemented in $\mathcal{O}(N)$ and have similar execution time. Furthermore, on Figure 5.5 we also observe that our bound is very close to the worst-case precision obtained using the exhaustive search for the first two hops. This last point is detailed in the following paragraphs.

Figure 5.6 represents the same results as previously but with a focus on the first two hops. For each hop, it shows the precision distribution obtained by simulations and depicts the upper and lower bounds of our approach. It presents as well the largest error observed during exhaustive search and the bound of Gutiérrez et al. [62]. In Figure 5.6, we observe that our upper and lower bounds are very close to the exhaustive search worst observation for the 2 first hops. Indeed, for the upper bound (respectively lower), we observe a difference with the exhaustive search of 5.4% (respectively 9.4%) for the first hop and 5.4% (respectively 9.9%) for the second hop. We also see

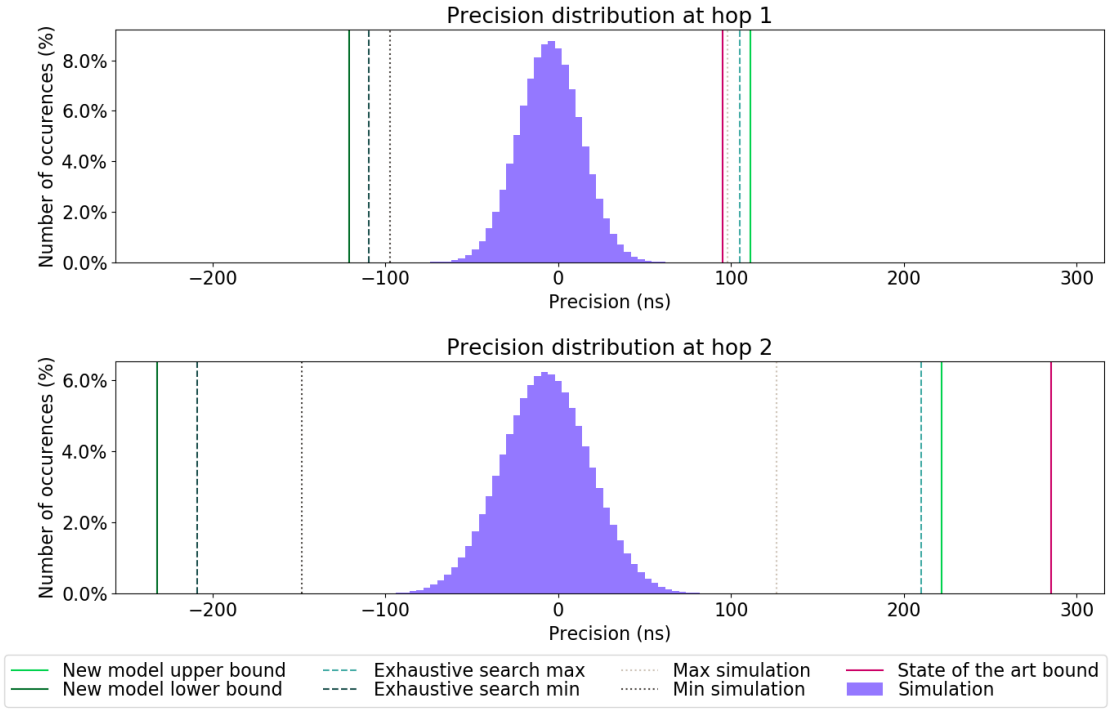


Figure 5.6: Comparison of simulated, exhaustive search and precision bound for 100Base-T physical layer on hop 1 and 2

Clock drift that leads to the	Hop #1	Hop #2	Hops #3
Minimum offset (in ppm)	-1.89	0.67	2.00
Maximum offset (in ppm)	-1.83	0.74	1.99

Table 5.3: Experimental clock deviation.

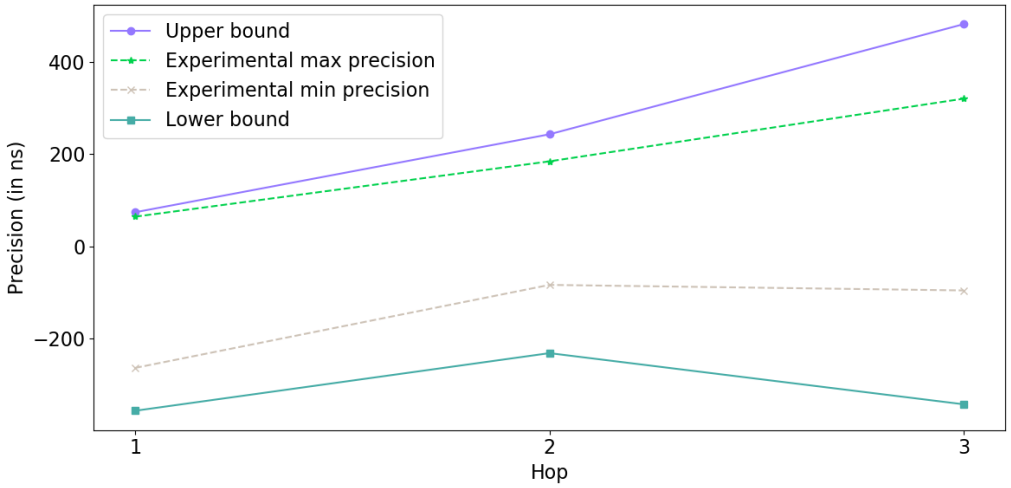


Figure 5.7: 100Base-T upper and lower bounds compared to measurements.

that the model of Gutièrrez et al. fails at the first hop, as it produces an upper bound which is smaller than the worst observed precision with the simulator and the exhaustive search. This is due to the asymmetries of 100Base-T which are not taken into account.

Experimental validation

Our experimental study is carried out with the data previously obtained in the experiments explained in the section 4.3.1 on the 3-hop chain of four Fraunhofer IPMS switches, where the first ones acted as Grandmaster. The progress of the clock was captured by a netTimeLogic PPS analyzer. The switches were in two-step mode, with a *pdelayInterval* of 1s and a *syncInterval* of 0.125ms.

To produce the results of Figure 5.7, 20 experiments of 1-hour of precision measurements have been made. Figure 5.7 presents largest and smallest precision records made over the 20 experiments at each hop. Between each 1 hour experiment, the interfaces were reset to allow measurements with different random combinations of asymmetry. Since no data traffic was transmitted during the experiments, similarly to the simulation study, we set J_{fup} equal to zero. To compare our upper and lower mathematical bounds with the measurements, we have to characterise the individual clock drifts of the time-aware systems, relatively to the Grandmaster clock rate. Thus, before each experiment, the free-running clock drift of the 3 time-aware systems were measured during 10 minutes. Using the PPS analyzer readings, we compute the drift rate between the Grandmaster clock rate and each of the three time-aware systems. Among the 20 values of clock rate deviation measured, we have kept the ones observed when the largest and smallest precision were recorded to feed our upper and lower bound models, respectively. Values measured are given in Table 5.3.

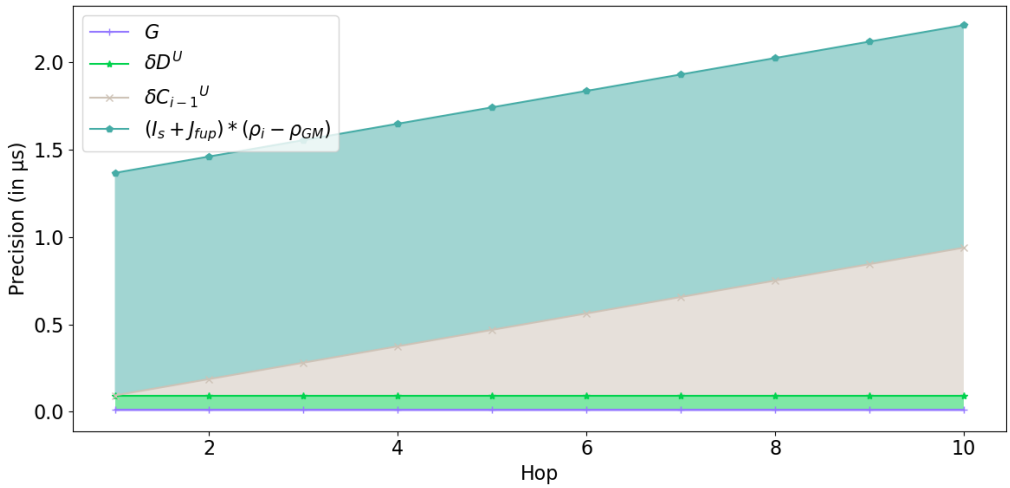


Figure 5.8: Impact of each term of Equation (5.23) on precision bound for 100Base-T.

Figure 5.7 compares the upper and lower bounds obtained with our worst-case model and the minimum and maximum precision values recorded during the 20 experiments. These results show that the worst-case model bounds nicely frame the actual measurements with little pessimism despite the small amount of measurements made. Moreover, as for the simulation, we observe that when the number of hops increases, it gets harder to measure worst-case events since the sequence of events that leads to the worst-case becomes less likely.

We did another campaign of 200 1-hour experiments for a one-hop network to increase our chances to approach the worst-case precision. In this experiment, the drift deviation of the slave time-aware system measured before each experiment is between -1.9 and -1.6ppm. Since this deviation is negative, we have to focus on the lower bound because the greatest difference between both clocks that will be observed will be negative. The smallest precision value recorded during the experiment is of -276ns. Using the drift deviation measured before the experiment that has led to the observation of the smallest precision, we obtain a bound on the worst-case of -332ns. We observe thus a very small difference of 56ns, which exhibits the reduced pessimism of our model in this case.

5.3.2 Analysis of the impact of parameters

Figure 5.8 highlights the impact of each term of Equation (5.23) on the worst-case precision bound as a function of the number of hops. Using this figure, we observe that, with the parameters of Table 5.2, the error is mainly caused by the drift that is proportional to the time between two synchronization procedures. This error can easily be reduced by increasing the frequency of the `Sync` and `Follow_Up` or by using higher quality clocks. So in this case and for hop 7, reducing the *syncInterval* from 125ms to 62.5ms improves the bound from 2.17μs to 1.54μs. However, even assuming that the clocks are nearly perfect, the bound remains of 0.92μs for the seventh hop. This is due to the physical jitter, the asymmetries and the granularity impacting the `Sync` network crossing duration and the accuracy of the *pdelay* and *correctionField* computation.

In order to better understand the components of the worst-case precision, we have studied the influence of each parameter on our upper bound. These results are presented in Figure 5.9. To

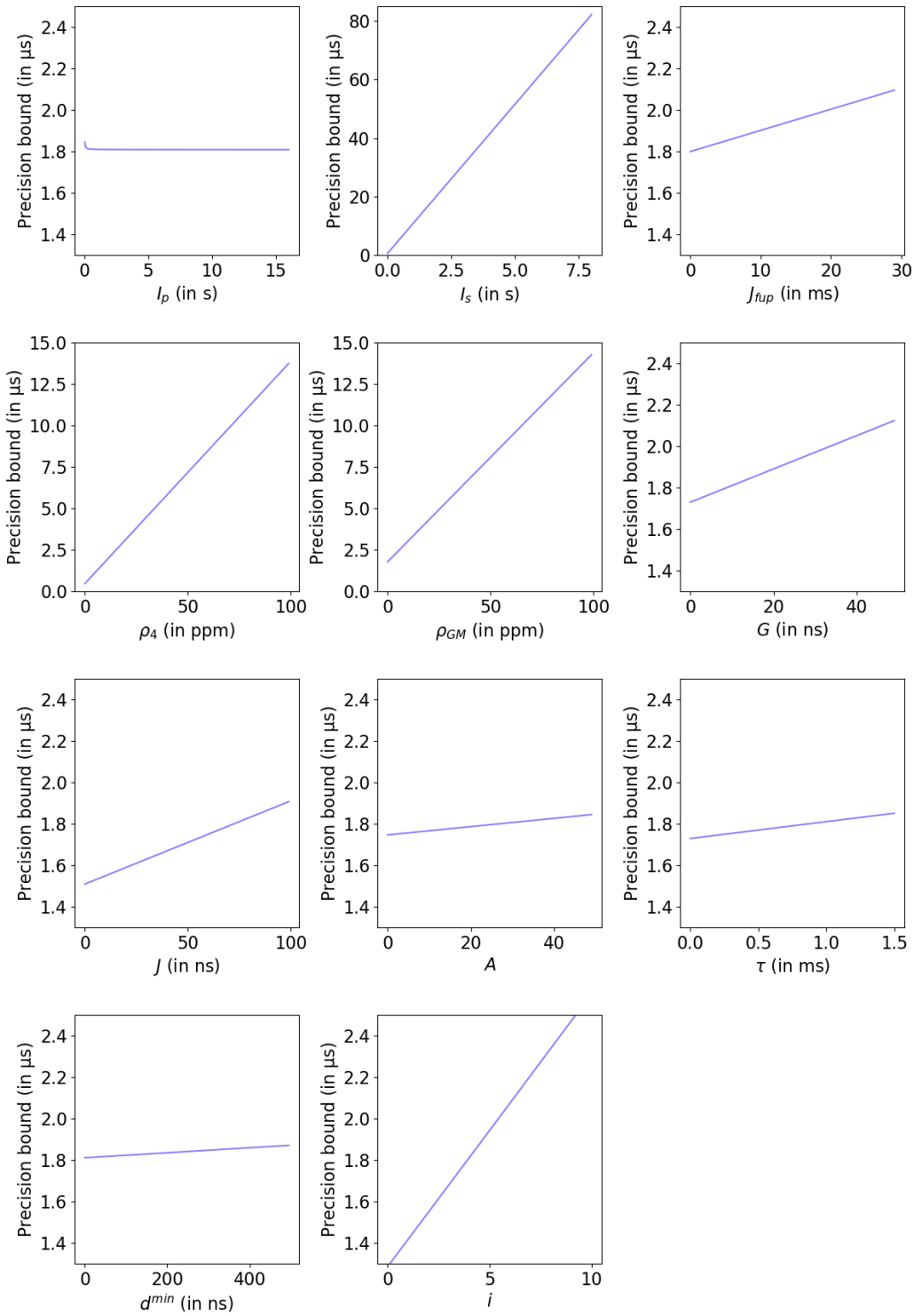


Figure 5.9: Impact of parameters on worst-case bound for 100Base-T physical layer

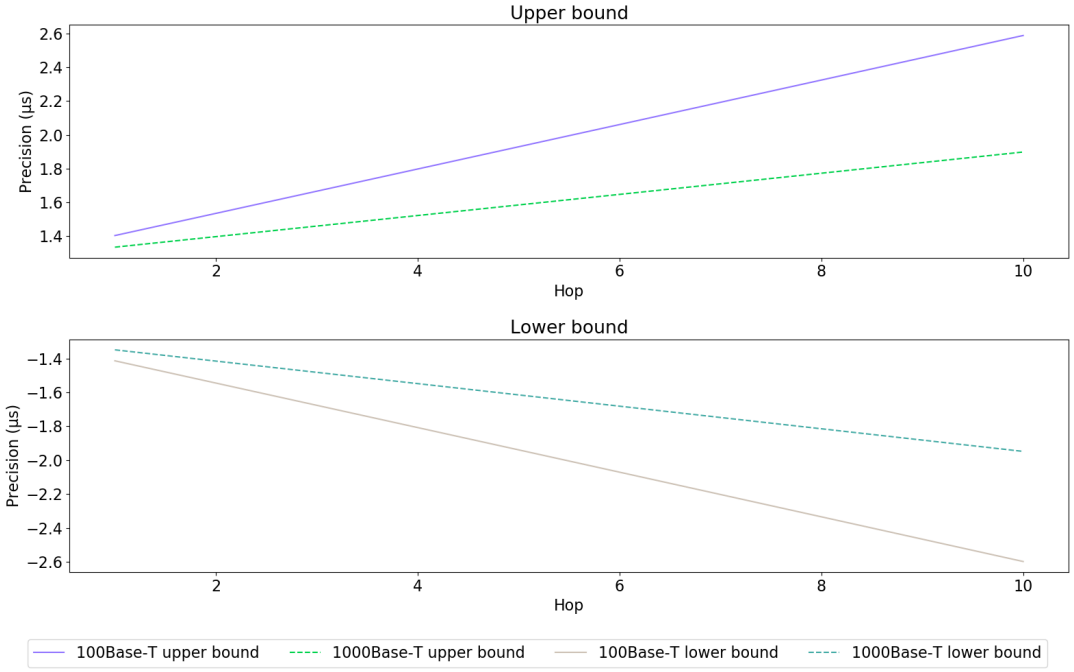


Figure 5.10: Comparison of 100Base-T and 1000Base-T precision bounds

produce these curves, we have used the parameters described above and set the number of hops to 4 ($i = 4$) and we made them vary independently.

In Figure 5.9, we observe that three parameters have a considerable impact on the bound: the *syncInterval* I_s, the clock quality of the Grandmaster ρ_{GM} and of the other time-aware systems ρ_i, and the number of hops i. Jitters experienced by `Follow_Up` J_{fup}, granularity G, physical jitter J, physical asymmetry A have a low impact but may help in reducing the bound for the cases where the limits on the *syncInterval* and the clock quality are reached. Other terms such as I_p, τ and $d^{\min}$ have a negligible impact on the bound and therefore it is not necessary to estimate them finely to carry out a worst-case analysis of the precision.

5.3.3 Comparing 1000Base-T to 100Base-T

For the 1000Base-T model instance, we use the same parameters as for the 100Base-T except for the parameters related to the physical layer given in Table 5.2. These values were derived from 4.4.

Similar results to 100Base-T are obtained for 1000Base-T when comparing simulations, exhaustive search and lower and upper bounds. For the upper bound, a difference of 20.1% (respectively 20.5%) is reached between the bound and the exhaustive search at 1 hop (respectively 2 hops). As for the lower bound, a difference of 17.3% (respectively 19.8%) is reached between the bound and the exhaustive search at 1 hop (respectively 2 hops). This greater difference is explained by the fact that the combination of jitter and granularity obtained for our switches in 1000Base-T does not allow us to meet the conditions described in the section 5.2 and reach the worst case. For example, we have observed that for the *neighborRateRatio*, the exhaustive search can't observe the condition that leads to the worst $t'_4 - t_4$ delay.

Figure 5.10 compares the upper and lower bounds obtained for 100Base-T and 1000Base-T instances for our TSN switches. In this figure, we observe that the use of 1000Base-T gives a more precise bound than 100Base-T: the 1000Base-T upper bound (respectively lower bound) is 36% (respectively 33%) smaller after 10 hops. This is due to the smaller physical jitter and asymmetries in the physical layer, thus reducing the worst-case error in the *Pdelay* mechanism (for the upper bound : $\delta D^U = 52.31ns$ for 1000Base-T compared to $\delta D^U = 121.06ns$ for 100Base-T). This result does not guarantee that 1000Base-T always offers better precision than 100Base-T with any time-aware system other than the hardware we have investigated in this study.

Conclusion

In this chapter, we propose a refined analytical model to upper or lower bound the precision of an IEEE802.1AS synchronization protocol implementation. These bounds rely on a generic communication model which captures link jitter and asymmetries. This communication model can be implemented for different Ethernet physical layer technologies using appropriate parameters. For 100Base-T links, we have shown that the upper bound on synchronization offers an almost two times reduction in pessimism with respect to the state-of-the-art solution and studied the factors impacting the precision. Intensive simulation and measurement, as well as exhaustive worst-case precision research, validate the reduced pessimism of our solution. The quality of our bounds comes as well from a refined characterization of the clock inaccuracies and of IEEE802.1AS protocol operations. In addition, we have also highlighted smaller bounds when using 1000Base-T compared to 100Base-T for the hardware we use.

During derivation, we assumed that all time-aware systems had the same characteristics in order to obtain a relatively comprehensible final formula. When this assumption is not true, an iterative calculation of the error at each hop is possible, as implemented in Python in Appendix B. A similar implementation has also been made in the commercially available network design and analysis software Timaeus-Net.

As for the future, this model is well suited to the very critical network but too safe for less critical ones. Indeed, if we take a step back and compare our bound to the distribution obtained with the simulation, we observe that although our bounds are close to the worst cases observed with the simulation, the latter are unlikely. For example, with 100Base-T (Figure 5.6), if we eliminate the 0.1% worst values, the pessimism between the bound and the worst simulation value is 51% for the upper bound and 47% for the lower bound at the first hop. This higher pessimism compared to our safe bound could lead to an oversize system if the need is a precision under a threshold x% of the time (in our example 99.9%). Furthermore, as mentioned previously, a filter could be applied to smooth the *Pdelay* value. Such filter will reduce the probability of appearance of the worst-case and increase the pessimism of our bounds if compared to experimental measurements with the device using filter. These two limitations of our safe approach could be solved by a probabilistic bound. Such a bound could be leverage in a less critical system like the airplane audio cabin use case.

Part III

Robustness

Chapter 6

Robustness mechanisms of IEEE802.1AS

As explained in the presentation of IEEE802.1AS, the standard offers two mechanisms to make synchronization robust to link, switch or Grandmaster failures. The first mechanism is the BMCA that deals with failure by reconfiguring dynamically the spanning tree and/or electing a new Grandmaster if needed. The second mechanism is a combination of external port configuration, multiple domains and hot standby mechanism.

In this chapter, we compare the two solutions in order to determine which one is the most suitable for critical embedded networks. We show that despite the similar performance of the two solutions on small networks, the static configuration is better suited to critical embedded networks, thanks to its simplicity and predictability. This work [40] was presented at the TSN/A 2022 conference.

6.1 Static and robust configuration

As previously explained, the use of multiple domains with hot-standby mechanism is not fully covered by IEEE802.1AS-2020. The P802.1ASdm amendment is therefore under discussion with the objective of proposing a standard in 2024. As a result of this incomplete standard, no silicon vendor offers a chip with this robust mechanism. In order to explore the implementation and the benefits of this mechanism, we worked in collaboration with Fraunhofer IPMS. This company sells a TSN FPGA switch IP with a gPTP software implementation. To address the missing points of the standard and which are not yet addressed in the P802.1ASdm drafts, we have proposed to the Fraunhofer IPMS team to implement very simple mechanisms to match safety and certification requirements for critical embedded networks. This software implementation has allowed us to carry out, to the best of our knowledge, the first robustness comparison experiments between the dynamic and static configuration while waiting for a standard mechanism.

The missing points of the current hot-standby standard are the following ones:

The failure detection procedure of a time-aware system. We propose to keep the same mechanism as for the BMCA. That is to say that a failure is detected after the non-reception of multiple consecutive `Sync` messages. As for the BMCA, this detection threshold is called *syncReceiptTimeout* and is set by default to 3.

The domain selection procedure after failure Another function which is not yet defined is the selection of the domain to use when several domains are active. We have decided to use the domains in the order of their ID. Thus if a failure is detected on domain 0 (no reception of three consecutive `Sync`), the time-aware system selects the domain with ID 1.

Synchronization quality evaluation of a domain. The function that evaluates whether a domain is of sufficient quality to be used is not yet fixed in the drafts of the amendment. This functions has not been implemented due to time constraints of the Fraunhofer IPMS team. Nevertheless, this function is less important in this first evaluation of the robustness mechanisms of IEEE802.1AS since we are not studying the impact of this kind of failure.

6.2 Investigated failures

Many failures can impact the synchronization process. In this manuscript, we will study only the following set :

- link failure
- switch or end-node failure
- Grandmaster failure

These three types of failure have an impact on the `Sync` and `Follow_Up` distribution tree. Indeed, the loss of a link or a switch can cut off access to the spanning tree for some time-aware systems. A Grandmaster failure simply causes the `Sync` and `Follow_Up` messages to disappear. To solve these three type of failures, it is necessary to reconfigure or change the spanning tree.

Other failures such as malfunctions in the execution of the protocol or the propagation of the false time base by a malfunctioning Grandmaster are not taken into account in this manuscript. Indeed, these failures being largely linked to the implementation of the protocol and the hardware used, we believe that a safety study specific to the considered hardware and software would make more sense.

6.3 Static versus dynamic configuration comparison

To compare the performance of the dynamic BMCA configuration to the one of the static hot-standby configuration with several domains, we use five qualitative and quantitative metrics. These metrics are chosen to match the needs of a critical embedded network deployment context. The metrics are:

- The first metric is failure detection and mitigation power. Indeed, since static and dynamic configuration are very different in the way they circumvent a failure, it is necessary to compare them.
- The second metric is the bandwidth consumed by the two mechanisms. Robustness of a static configuration is a function of the number of domains. The more domains, the more traffic is needed for keeping them in sync. So we'll compare the bandwidth consumed by the two mechanisms for different network and mechanism settings.
- Third, we'll study the reconfiguration time in the event of a failure. Indeed, this time is a key criterion for oscillator drift characterization. But above all, this reconfiguration time needs to be deterministic and bounded for safe use of gPTP in a critical network.

- Fourth, we will study the impact of reconfiguration time on synchronization precision. Indeed the larger the loss of precision during reconfiguration, the higher the impact on synchronization users, i.e. TAS and/or applications.
- Finally, we'll discuss the design complexity of static configuration versus dynamic and plug and play configuration.

To evaluate the quantitative metrics, we will use a series of experiments on a test bench, a study of the bandwidth used as well as the model to determine bounds on the worst-case precision in the event of reconfiguration. For qualitative metrics, we will use information from the standard and our experience on small networks.

6.3.1 Failure detection and mitigation power

For this comparison, we use the same failure detection mechanism for both methods. In case of non-reception of three consecutive `Sync` messages, the protocol takes corrective actions. In the dynamic case, a new tree is built using `Announce` messages and BMCA algorithm. In the static case, a new domain is selected among those available. Thus, both mechanisms detect the failure at the same time.

From the point of view of mitigation power, BMCA is a very powerful mechanism. Indeed, as long as there is a path to a time-aware system, `Announce` messages will find it and allow the creation of a new spanning tree thanks to message broadcasting. In case of Grandmaster failure, the BMCA even allows the election of a new GM and the creation of the spanning tree which allows the distribution of `Sync` messages.

In contrast, the static multiple domain configuration is limited by the number of domains and Grandmasters configured statically. In the case where our network only has one Grandmaster and two spanning trees as shown in Figure 6.1, a single failure of the Grandmaster, the first link or the first switch in the spanning tree leads to the total loss of synchronization, whereas the BMCA would have elected a new GM. It would therefore take an important number of Grandmasters and domains in a static configuration to match the mitigating performance of BMCA on any network.

However, in an embedded and especially critical context, the differences between the two mechanisms are more limited. Indeed, although the BMCA can select the best Grandmaster available, the set of devices having access to a GPS antenna to retrieve the time or having a good quality oscillator to remain stable in freeruning will be limited due to complexity and cost reasons. Thus, in the use cases studied in the EDEN project, only 2 to 3 Grandmasters are envisioned per network. In addition, the topologies used generally offer no more than two to three independent paths to contact each time-aware system. It is therefore possible to match the mitigation power of BMCA with the static configuration with two to three domains per Grandmaster and offer the same level of robustness as data communication.

Figure 6.2 illustrates the statements of this paragraph within the scope of a size-modulable ring topology envisaged to connect time-aware systems in the cabin of an aircraft such as lights, buttons, ventilation and loudspeakers. With this topology, two Grandmasters with two domains each provide resilience to at least one link, switch or Grandmaster failure. On such topology, more than two link or switch failures lead to the separation of the network into two parts and therefore no longer allows the transport of data between the two parts of the network. Synchronization is therefore no longer necessary in the disconnected part. Indeed, this topology allows the existence of only two independent spanning trees, enabling it to withstand only one failure. In other words, there is no robustness gain in adding another Grandmaster or spanning tree on such a topology. The notion of spanning tree independence and the mitigating power of such a set of spanning trees will be discussed in detail in the next chapter.

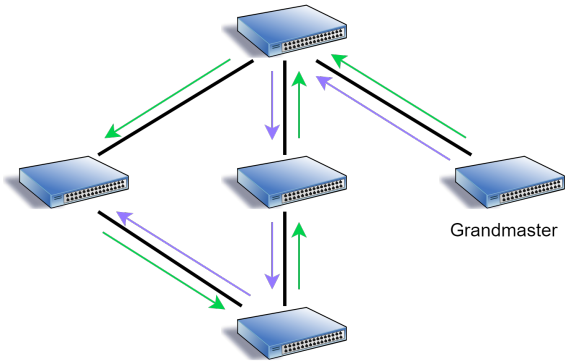


Figure 6.1: Illustration presenting a simplified version of an automotive network with one Grandmaster and two domains.

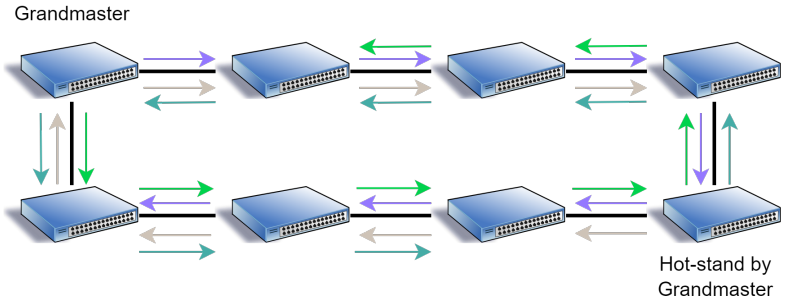


Figure 6.2: Illustration presenting a cabin airplane network with two Grandmasters and four domains.

	Sync	Follow_Up	Pdelay_Req	Pdelay_Resp	Pdelay_Resp _Follow_Up	Announce
Size (Bytes)	64	94	72	72	72	68 + 8 * hop_count

Table 6.1: AS frame size in bytes with Ethernet header

	BMCA	Static configuration between two switches	Static configuration between switch and end-station
Bandwidth usage (%)	0.00130	0.00217	0.00417

Table 6.2: Bandwidth consumption depending on the robustness mechanism used for different locations in the network

6.3.2 Bandwidth usage

One of the effects induced by the use of several domains is the multiplication of Sync/Follow_Up messages per domain. Thus in this subsection we quantify the bandwidth usage of both mechanisms.

For this evaluation, the BMCA and its static concurrent use the same messages at the same recurrence with the following exception. With the static configuration, the use of message Announce is unnecessary. IEEE802.1AS-2020 does not allow Announce messages to be disabled to remain compatible with IEEE1588. However, other standards like AUTOSAR specify that in case of BMCA deactivation, no Announce message should be sent. As this standard limitation should be discussed and reworked by the IEEE1588 and TSN working groups, we consider disabling Announce messages when using static robustness mechanisms.

When using the BMCA, bandwidth consumption ratioB_{BMCA} can be expressed as follows:

$$B_{BMCA} = [N_{Sync} \times (S_{Sync} + S_{fup}) + N_{pdelay} \times (S_{pReq} + S_{pResp} + S_{pRespFup}) + N_{Announce} \times S_{Announce}]/d \quad (6.1)$$

Where $N_{message}$ is the number of message per second, $S_{message}$ is the message size and d the link rate. The size of the different messages used by the protocol is given in Table 6.1. Only the size of the Announce message is dependent on the number of hops because it includes information on the traversed time-aware systems. We consider in the rest of this section that the Announce message crosses at most ten devices, as it is representative of the maximum size of the embedded networks studied during this thesis.

Meanwhile, the bandwidth consumption ratio when using the B_{static} multi-domain static configuration on the link l can be expressed as follows:

$$B_{static}(l) = \left[\left[\sum_{j=0}^{N_{dom}} N_{Sync}(j,l) \times (S_{Sync} + S_{fup}) \right] + N_{pdelay} \times (S_{pReq} + S_{pResp} + S_{pRespFup}) \right] /d \quad (6.2)$$

Where N_{dom} is the number of domain, $N_{message}(id, l)$ is the number of message per second for a specific domain id on the link l, $S_{message}$ is the message size and d the link rate. this equation assumes that the Common Mean Link Delay Service (CMLDS) is used. This means that the single *pdelay* exchange is shared by all domains.

The results obtained on the topology of the Figure 6.2 with 1Gb/s link and default standard values for the number of messages per second are presented in Table 6.2. First, regardless of the

mechanism used, the AS frame size being relatively small, the bandwidth consumed remains very low with less than 0.005% of a 1Gb/s link and less than 0.05% if we use 100Mb/s link. Secondly, we observe that the bandwidth consumed is greater when using several domains than with BMCA due to the multiplication of the number of `Sync` and `Follow_Up` exchanges. In this example, using CMLDS saves 0.0005% of bandwidth. Moreover, despite the use of 4 domains in our example, only two domains cross a core network link in the same direction, thus limiting the increase in bandwidth consumption. This effect is induced by the search for independent spanning trees in order to offer the best robustness to failure.

As for a switch/end-station link, the `Sync` and `Follow_Up` exchange of a domain always push the synchronization to the end-station. Thus, traffic is multiplied by the number of domains on these links compared to the BMCA.

One point that is not highlighted by the results of Table 6.2 is the impact of the transient state observable with the BMCA. Indeed, during a reconfiguration period, several `Announce` messages coming from different Grandmasters cohabit. Thus, for our example, the consumed bandwidth increases by 0.00013% per Grandmaster capable device during this transient period.

6.3.3 Reconfiguration duration

One of the very strong needs of the critical embedded world is to ensure that the synchronization reconfiguration time is bounded. In this section, we study this reconfiguration time for the two mechanisms. For this, we will use the two topologies presented in Figure 6.3. The experimental setup for the 3-switch topology is described in the Figure 6.4. The experimental setup for the other topology is similar and contains only one additional UART link between switch 3 and the control computer. The experience is as follows:

- Switches are turned on and configured;
- Synchronization service is started and we wait for all switches to be synchronized;
- Network operates normally for a random time between five and six seconds;
- A failure is triggered by disabling gPTP on one of the ports of the faulty link;
- The fail-over procedure is started either by choosing a new domain for the static configuration or by resetting the state of the ports using the BMCA;
- Switches are turned off.

These steps are repeated 1000 times per experience. The switches used are the FPGA-based TSN switch from Fraunhofer IPMS and are controlled by a UART link. We will call hereafter reconfiguration time, the duration between the creation of the failure via a UART command and the information message of return to operational condition which is sent by the switch on the UART link. These writes and reads on the UART link are done by a python script. The AS configuration used for these series of measurements is the default one (i.e. $syncInterval = 125$ms and $syncReceiptTimeout = 3$). Next we present the measurement results for the BMCA first, then for the static reconfiguration (domain change) and for the hot standby static case which leverages the presence of the backup Grandmaster.

BMCA reconfiguration duration. The expected stable configurations before and after BMCA reconfiguration is given in Figure 6.5. The results obtained by measuring the reconfiguration time on Switch 2 for both topologies are presented in Figure 6.7. For the 3-switch (respectively 4-switch) topology, we observe a configuration time between 282ms and 541ms (respectively between 281ms and 539ms). The minimum value is close to the expected one. Indeed, the `Sync` being sent every

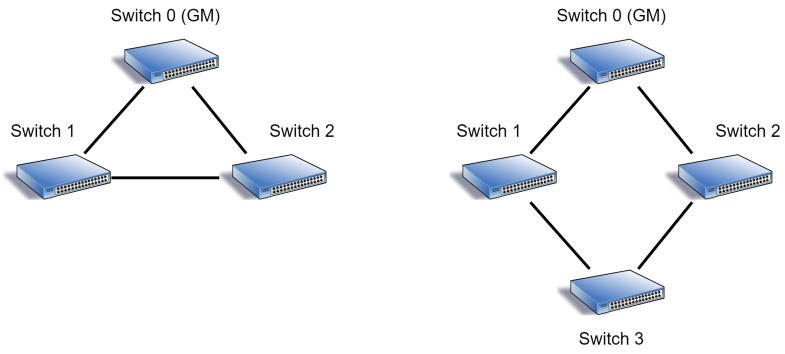


Figure 6.3: Topologies used to study the reconfiguration time of the two robustness mechanisms of IEEE802.1AS

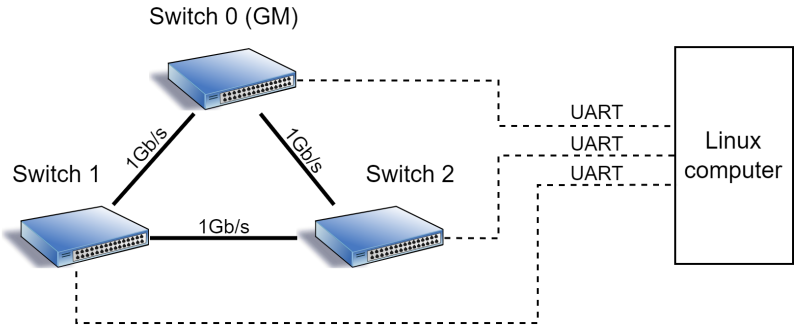


Figure 6.4: Experimental setup for the 3-switch topology experimentation.

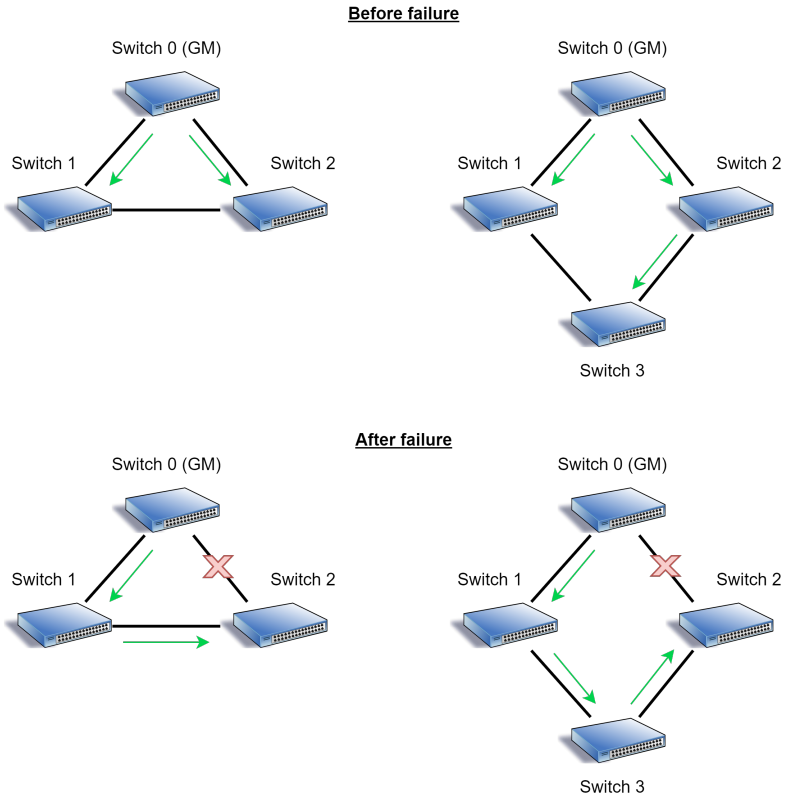


Figure 6.5: Expected configurations before and after failure with BMCA for the two topologies.

125ms and the failures being detected after the loss of three `Sync`, the failure can be at best detected after 250ms, as shown in Figure 6.6 - Top. The maximum value is also close to the expected value of 500ms. Indeed, in case of a failure just after the reception of a `Sync`, a failure will be detected and thus trigger the execution of the BMCA in 375ms. To which is added up to 125ms to receive a new `Sync` via the new spanning tree drawn by the BMCA, as shown in Figure 6.6 - Bottom. The difference with the expected results is explained by a combination of latency caused by the non-real time side of the UART link, the python measurement script and the latency of the BMCA in the switch execution. A more precise method using the FPGA GPIOs and an oscilloscope could not be implemented due to the lack of access to the source code.

We also observe a difference in probability in the reconfiguration time depending on the topology. Indeed, a long reconfiguration time (between 410ms and 541ms) is more regularly measured with the 3-switch topology. On the other hand, a shorter reconfiguration time (between 281ms and 410ms) is more likely with the 4-switch topology. This difference in probability of occurrence seems to be linked to the order of detection of the failure by the different switches. Indeed, with the 3-switch topology, switch 2 sometimes has time to become Grandmaster after detecting the failure. It then starts sending `Announce`, `Sync` and `Follow_Up` messages before receiving an `Announce` message from switch 0, enabling it to execute the BMCA to reach the expected state. But, less regularly, switch 2 receives information from switch 0 via switch 1 before it has time to try to become Grandmaster, leading to the lower reconfiguration time observed in Figure 6.5. With the 4-switch topology, the

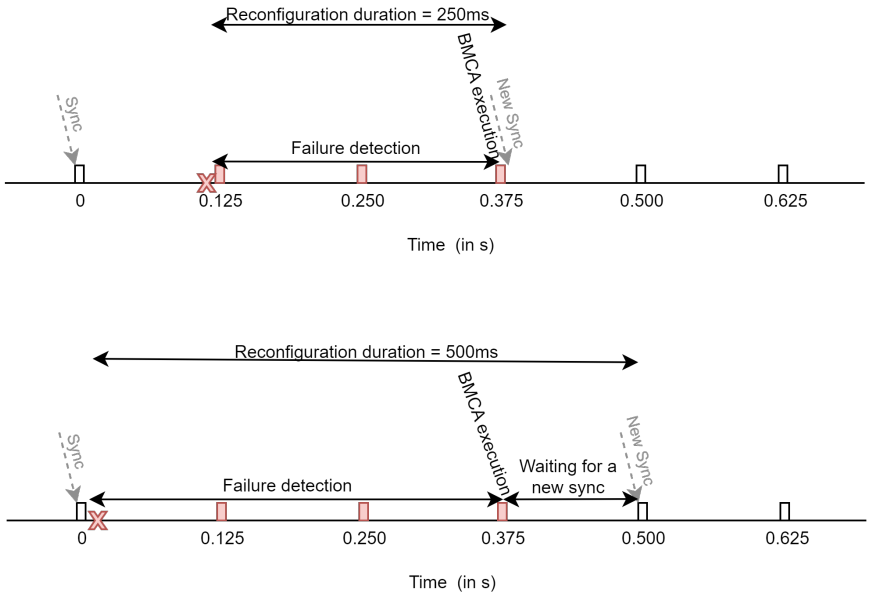


Figure 6.6: Diagram showing the sequence of events leading to the smallest (top) and largest (bottom) reconfiguration time when using the BMCA

opposite behavior is observed. Switch 2 has less opportunities to pass Grandmaster before realizing that switch 0 is still accessible via the other path. However we don't really know how to explain this transient state in details without more measurement. This raises the question of the explicability and determinism of the reconfiguration time obtained with this mechanism, especially in the case of networks more complex than a ring.

Static domain reconfiguration duration. The second set of measurements evaluates the static configuration, whose expected behaviour is described in the Figure 6.8. The distribution of the switch 2 reconfiguration times is presented in the Figure 6.9. Our measurements show a very close reconfiguration time for the two different topologies between 252 ms and 399ms. For these two topologies, the expected reconfiguration duration is between 250 and 375ms. Indeed, the reconfiguration is possible as soon as the failure is detected because the time-aware system is already synchronized to the backup domain. As for the previous experiment, we observe a slight overhead caused by the non-real time behaviour of the instrumentation system. However, here the minimum reconfiguration time is 30ms lower than for the BMCA. We can therefore assign at least part of this difference to the execution of the BMCA while keeping in mind that this switch is prototype.

Static hot standby reconfiguration duration. The last set of measurements studies the reconfiguration time with a static configuration using the hot standby mechanism in the case of a Grandmaster failure. The expected behaviour is presented in Figure 6.10 and the distributions obtained in Figure 6.11. As for the previous measurements, the distribution obtained does not vary with the topology. The values obtained are between 253ms and 412ms. We assume that the higher maximum reconfiguration time than the simple static configuration is caused by a different implementation by the switch of the change between domains of the same Grandmaster and the change

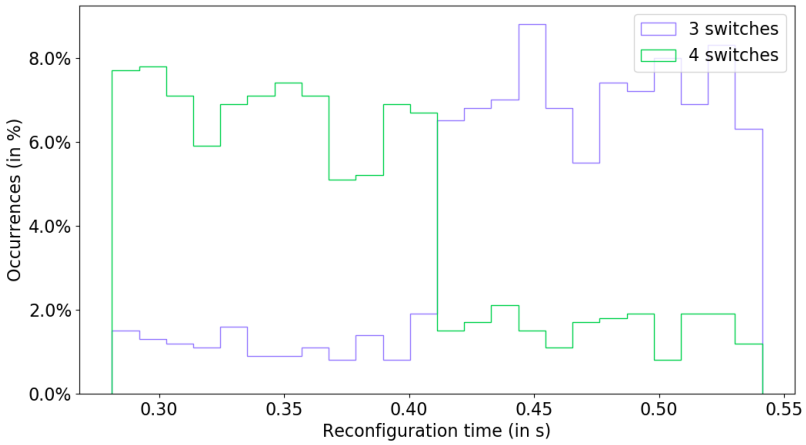


Figure 6.7: Distribution of the reconfiguration time after a link failure with BMCA on the 3- and 4-switch topologies

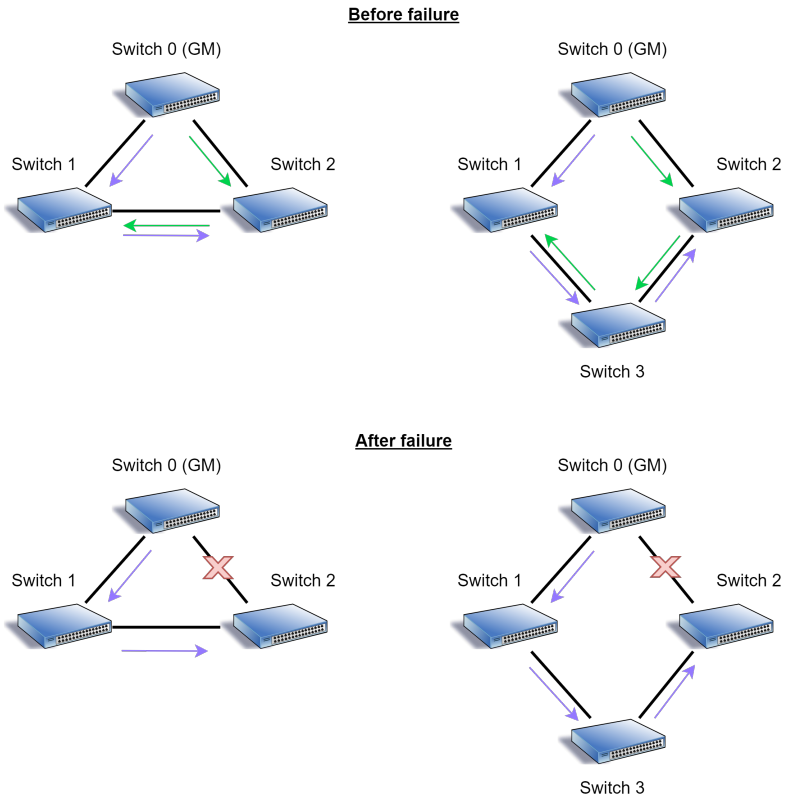


Figure 6.8: Expected static configuration before and after failure for the two topologies. (Green : primary domain; Purple : backup domain)

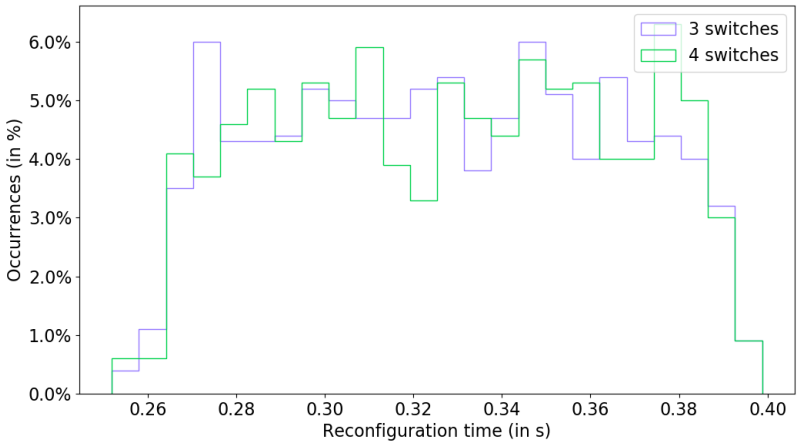


Figure 6.9: Distribution of the reconfiguration time after a link failure with two static domains on the 3- and 4-switch topologies

between domains of a different Grandmaster.

These three series of measurements show a faster reconfiguration time, easily predictable and especially not varying according to the topology and the location of the failure when using the static robustness mechanism compared to the BMCA. Even though further investigations are needed to study larger networks, the hypothesis of a constant reconfiguration time of the static mechanisms and of a variable configuration duration for its dynamic counterpart is confirmed in these experiments.

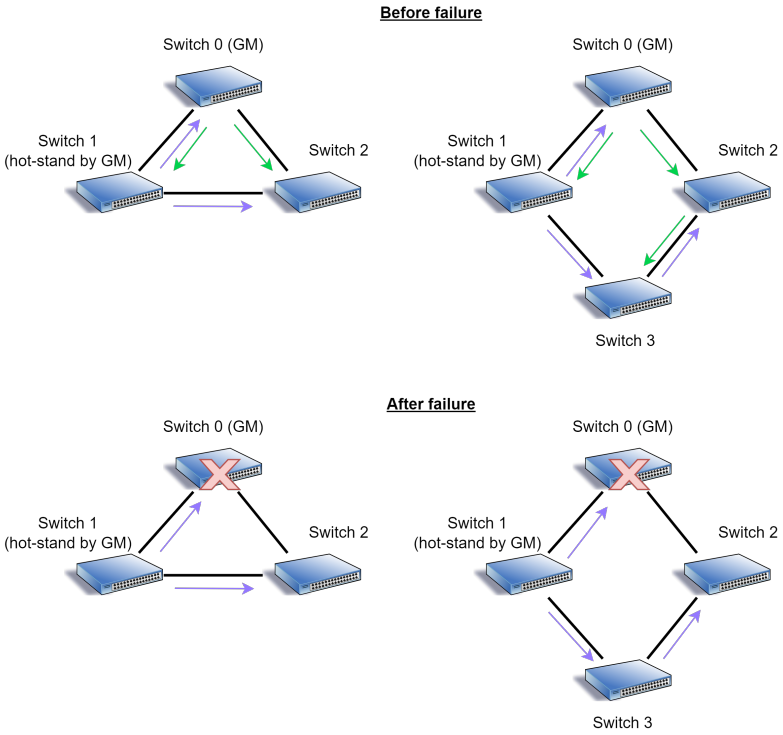


Figure 6.10: Expected static configuration with hot standby GM before and after failure for the two topologies (Green : primary domain; Purple : backup domain)

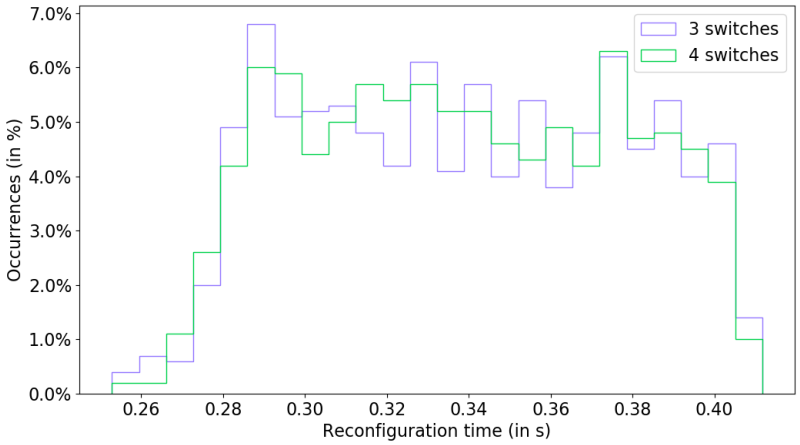


Figure 6.11: Distribution of the reconfiguration time after a Grandmaster failure with hot standby Grandmaster for the 3- and 4-switch topologies

6.3.4 Impact on the synchronization precision

The longer the reconfiguration time, the more time the clock has to drift. This induces a worse precision in case of failure. In this section, we study the impact of failures with the two mechanisms on the precision using the reconfiguration times measured in the previous section and the model to compute a safe bound on the precision described in chapter 5. For this comparison, we assign to the parameter I, which represents the duration between two resynchronizations in the equations used to determine the bounds on precision, the maximum reconfiguration time measured in the previous section (i.e. $I = 541ms$ for dynamic robustness and $I = 412ms$ for static robustness). We will only display the results of the upper bound in this subsection because of the small difference between lower and upper bounds.

To compute this bound, we use the 1Gb/s link bound model with the same parameters that were used in the Chapter 5 and that are recalled in Table 6.3. The bounds obtained for the different topologies are presented in Table 6.4.

As expected, the longer reconfiguration time of the BMCA induces a larger bound on worst case precision than for the static configuration. In the case of our 3-switch and 4-switch topology, a difference of 1.293ms is observed caused by the additional 129 ms of reconfiguration duration. This higher BMCA bound can be easily compensated by using better quality oscillators at a higher cost or by using a lower *syncInterval*, thus increasing the frequency of the `Sync` and `Follow_Up` messages at the cost of higher bandwidth consumption. For example, using the next smallest syncInterval (i.e. 62.5ms) and transposing the reasoning explaining the measured reconfiguration time of the BMCA in the previous subsection, the reconfiguration time obtained is then 291ms. With such a time, the bound is then 3.103µs for the 4-switch topology. This bound is smaller than the one obtained with the static configuration for a syncInterval of 125ms at the cost of a bandwidth consumption of 0.00230% of a 1Gb/s link, i.e. a higher consumption than on a switch/switch link with the static configuration studied in subsection 6.3.2.

6.3.5 Complexity in the design phase

The BMCA, being a plug and play solution, only provides to the network designer two parameters to regulate the `Sync` messages distribution : *priority1* and *priority2*. As explained above, these two parameters allow a time-aware system to choose a Grandmaster over another one or completely

Parameters	Value
$syncInterval$	125ms
$pdelayInterval$	1s
ρ_{GM}	0.02ppm
ρ_{Slave}	10ppm
G	10ns
τ	1ms
d^{min}	200ns
$J_{i \to j}$	29.7ns
$J_{j \to i}$	8ns
A	6.85ns

Table 6.3: Parameter for the precision bound calculation using the model of Chapter 5 in case of failure.

Configuration	Upper bound (in µs)
3-switch topology with BMCA	5.546
3-switch topology with static configuration	4.253
4-switch topology with BMCA	5.608
4-switch topology with static configuration	4.315

Table 6.4: Bound on the precision in case of failure for different topologies and robustness mechanisms using the model of Chapter 5

prevent a time-aware system from being elected Grandmaster (priority1 to 255 prevents sending `Announce` messages). By playing with *priority1*, it is therefore possible to select the Grandmaster that can be used and in what order.

The static multiple domain configuration gives the network designer full control over the synchronization trees. Such a configuration contains all the gPTP port static states, equivalent to a spanning tree, for each domain. However, this total control opens up new problems such as:

- How to find a precise configuration?
- How to find a robust configuration?
- What is the optimal configuration?
- How many failures can my configuration withstand?
- Where to place hot standby Grandmasters in the network?
- What is the impact of increased robustness on precision?
- Does this configuration allow the appearance of a concurrent time base in the event of a failure?

These questions will lead to a more complicated and lengthier design phase than for the BMCA.

Conclusion

Although this study was conducted with assumptions about the content of the IEEE802.1ASdm amendment, it allows us to study the differences between the two robustness mechanisms proposed by the IEEE802.1AS standard.

The BMCA being a distributed and dynamic solution is consequently better suited to a constantly evolving network such as the ones deployed in robotized assembly line for instance. This solution is very easy to use and offers the best adaptability thanks to its broadcast messages that discover synchronisation paths. However, this dynamism is the cause of a slower reconfiguration. Indeed, after the detection of the failure, message exchanges are necessary before being able to synchronize using a new spanning tree. Moreover, a transient state can be observed on larger topologies, leading to successive synchronization at several time bases before stabilization. This transient state raises the still unstudied question of the determinism and the possibility to bound the reconfiguration time of this mechanism.

As for the static solution, it offers a mechanism that provides a faster and deterministic reconfiguration time at the cost of higher bandwidth usage, limited adaptability to failure due to finite number of domains and Grandmasters configured, and a much higher complexity in the network design stage. However, these counterparts are very limited in the case of critical embedded networks. Indeed, the latter rarely offer more than three independent paths to contact the different devices. Thus, a small number of domains and Grandmasters is sufficient to provide a level of adaptability to the failure equivalent to the dynamic solution. This low number of domains also limits the explosion of the bandwidth usage, which remains higher than the one needed by the use of BMCA while

remaining well below 0.1% of the bandwidth even with 100Mb/s links. As for the complexity of the design phase, the world of critical embedded systems is used to it because it simplify the certification effort. For example, complex design phases can be found in AFDX network dimensioning with validation using network calculus based analysis or in critical calculator with WCET calculation or interference analysis.

Although offering similar performance on our small test network, the static configuration is more suitable for critical embedded networks because of the question of determinism and the reconfiguration time. To offer the same level precision guarantee with BMCA, a reduced synchronization interval or an over-sizing of the oscillator's quality has to be chosen for the small network configurations investigated in this chapter. However, for more complex networks, it is not clear yet if the BMCA can provide a deterministic reconfiguration duration or if we can derive a bound on this duration that is safe. Thus, industrial partners consider static configuration a much safer way to go.

Chapter 7

Design of a static configuration

As highlighted in the previous chapter, static robustness mechanism is better suited for critical embedded networks at the cost of manual configuration, which raises issues that are not encountered by its dynamic counterpart. Indeed, the static configuration leaves the decision power to the network designer. This designer must choose the state(Master, Slave or Passive) of each gPTP port for each domain, in other words design a spanning tree per domain, and then make sure that the configuration he proposes is robust but also precise and that it can withstand a predefined number of failures. In this chapter we propose a method for determining the most precise configuration among the most robust ones. To do so, we begin by looking at different metrics for evaluating precision and robustness. Next, we focus on the relationship between precision and robustness. Then, we extend the study to "classic" and hot standby multi-Grandmaster configurations and the problems that such configurations raise. Following, we study the time complexity of the algorithms proposed in the previous sections and propose optimisations before presenting the complete methodology. Finally, we propose ways of optimising the network in order to improve the precision and/or robustness of the synchronisation using metrics. In this chapter, the results will be discussed using a subset of our use case topologies but have been tested on all the EDEN project case studies. This work [42] was presented at the E&IP@ATD 2023 conference.

7.1 Configuration performance metrics

First, we need to evaluate the precision and the robustness of a configuration in order to compare them. This evaluation is based on the use of different metrics. Several metrics will be proposed. Since these metrics will be leveraged to pick configurations that give an appropriate performance trade-off, we need to design metrics that are fast to compute. Discriminating power of metrics will be studied in order to select only one of them per quantity to be evaluated. As mentioned previously, all the metrics discussed in this section have been tested on all the use cases of Chapter 1, but only a subset of these results are presented in this section to illustrate the findings.

7.1.1 A precise configuration

Precision has been extensively studied in Part II of the manuscript. What can be retained for this chapter is that with identical hardware, the parameter impacting the most the precision of a configuration is the number of hops. Thus, to get a simple, easy to compute, precision performance metric, we leverage the number of hops that separate the Grandmaster from a time-aware system i

to represent the synchronization quality. It captures the fact that the further the node is from the Grandmaster, the worse its synchronisation precision gets.

We recall that a configuration can be model by a spanning tree rooted at the Grandmaster node. To measure the precision of a spanning tree x, we define a metric $M_{Dist}(x)$ that sums the distances between the Grandmaster and each time-aware system. As said, these distances are a naïve but simple precision quality indicator. The metric is defined as follows:

$$M_{Dist}(x) = \sum_{i=0}^{n} importance_i * distGM_i(x) \tag{7.1}$$

Where n is the number of time-aware systems in the spanning tree x, $importance_i$ is an importance factor of the time-aware system i and $distGm_i$ is the number of hops between the Grandmaster and the time-aware system i with the spanning tree x.

All the metrics we will see in this chapter incorporate a notion of importance. In the case of this metric, this term of importance aims at favouring a spanning tree where important time-aware systems are close to the Grandmaster. For example, a core network switch and a critical end station can be given a higher importance than less critical end stations. However, although the industrial partners of the EDEN project found it interesting, they could not give us representative values. For the following and in all our metrics, we will always use an importance equal to 1 for all time-aware systems.

By minimizing this metric, we obtain spanning trees where the nodes are on average closer to the Grandmaster. Thus, for identical hardware, we obtain the spanning trees where the average precision is the lowest, as illustrated with the spanning trees in Figure 7.1. On this example, the scores obtained are 20, 12 and 10 by going through the spanning trees from left to right. Spanning tree 3 obtains the best score since its maximum depth is 3 and its average depth is 1.7 while spanning tree 1 (respectively 2) reaches a maximum depth of 5 (respectively 3) and an average depth of 3.3 (respectively 2). In terms of worst-case precision, the maximum depth reflects the worst observable precision in the entire network. As for the average depth, it reflects the average worst case precision in the network. Thus, by minimizing the metric, we reduce the average precision and worst-case precision for the network. Moreover, we also observe that a better score on the metric does not always imply a better precision on all the nodes. Indeed, nodes 3 and 5 are closer to the root, and thus would benefit from a better precision, with the spanning tree 1 than with the spanning tree 2, although the metric score is lower for the spanning tree 2 than for the spanning tree 1. Furthermore, it is important to point out that the highest bound on precision in the entire network, using the same parameters as in the previous chapter, with spanning tree 1 is 1.584µs compared to 1.459µs for spanning tree 3, i.e. a difference of only 125ns between the worst and the best of the three spanning trees in terms of precision. Thus, a bad score does not always imply a significant difference in worst case precision.

We also considered the use of another metric reflecting the quality of the clocks on the path to all time-aware systems. Indeed, we assume that a lower quality time-aware system has a greater impact on other time-aware systems if it is close to the Grandmaster. To study this impact, let's consider three configurations of the same spanning tree branch composed of 10 time-aware systems having the same characteristics, as detailed in Table 7.1, except for the time-aware system 1 in configuration 2 and time-aware system 8 in configuration 3 having a worse quality clock (i.e. 50ppm) as shown in Figure 7.2. The objective is to compare the worst case bound on the precision of the time-aware system 9 for the three configurations. Using the model described in the chapter 5, the upper bounds for configuration 1 can be directly calculated but for configuration 2 and 3, some modifications are necessary to the model. Indeed, when deriving the *rateRatio* r and the *correctionField* C, we made

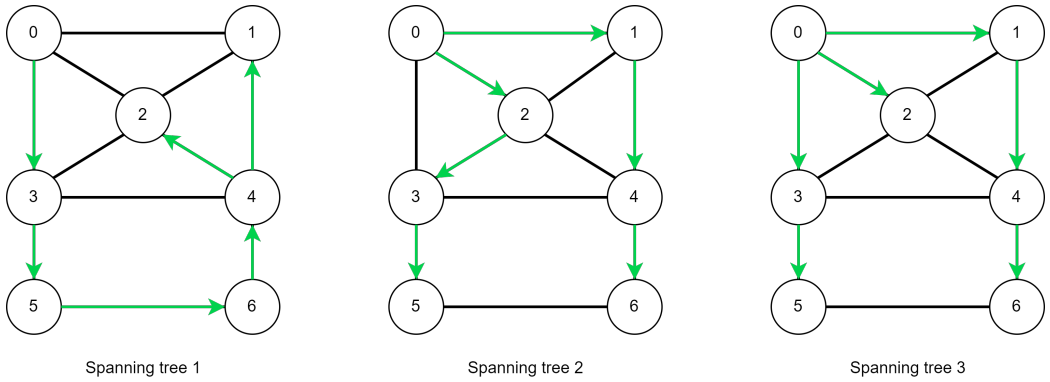


Figure 7.1: Three different spanning trees with node 0 as root on the A350 AFDX core topology.

G	ρ	d^{min}	$J_{j\to i}$	$J_{i\to j}$	A	τ	J_{fup}
10ns	10ppm	200ns	29.7ns	8ns	6.85ns	1ms	2ms

Table 7.1: Placement metrics evaluation parameters (from Fraunhofer TSN switch 1000Base-T characterisation).

the assumption that the time-aware system used were identical, which is not the case here. Thus we use the iterative implementation, described in Appendix B, that iteratively computes the errors for the different mechanisms at each hop to determine the worst-case precision over the whole chain.

Tables 7.2, 7.3 and 7.4 give the upper bound on the worst-case precision and computation details at each hop for the three configurations. Graph 7.3 shows the upper bound on the precision of the three configurations as a function of the number of hops.

Using the tables, the first observation from these calculations is the very small difference in the bound on the error of the *neighborRateRatio* and the *rateRatio*. Indeed, the difference between the value of configuration 1 and configuration 2 or 3 is at most 3.5×10^{-12} for δnr_i and 3.7×10^{-11} for δr_i at the same hop. The main difference is caused by the *pdelay* mechanism. Indeed, we calculate a bound on this mechanism 40ns larger (+77%) when the node of lower quality is requester or responder. This increase is caused by the greater drift that takes place during the exchange, the main factor of which is the processing time between receiving the `Pdelay_Req` and sending the `Pdelay_Resp`. This increase, although only occurring on the hop before and the hop after the imprecise node, is propagated to all the following nodes via the *correctionField* which leads to a difference between the upper precision bounds of 80ns after passing this node compared to configuration 1. This difference can be seen in the Figure 7.3 between configuration 2 and

Node	1	2	3	4	5	6	7	8	9
δr_i ($\times 10^{-7}$))	0.497	0.994	1.49	1.99	2.49	2.98	3.48	3.98	4.47
δC_i (ns)	62.36	124.76	187.22	249.73	312.29	374.9	437.57	500.28	563.05
δGM_i (ns)	62.31	124.67	187.07	249.53	312.04	374.6	437.21	499.87	562.59
P_i^U (µs)	2.562	2.625	2.687	2.750	2.812	2.875	2.937	3	3.063

Table 7.2: δr_i, δC_i, δGM_i and P_i^U of the nodes of configuration 1

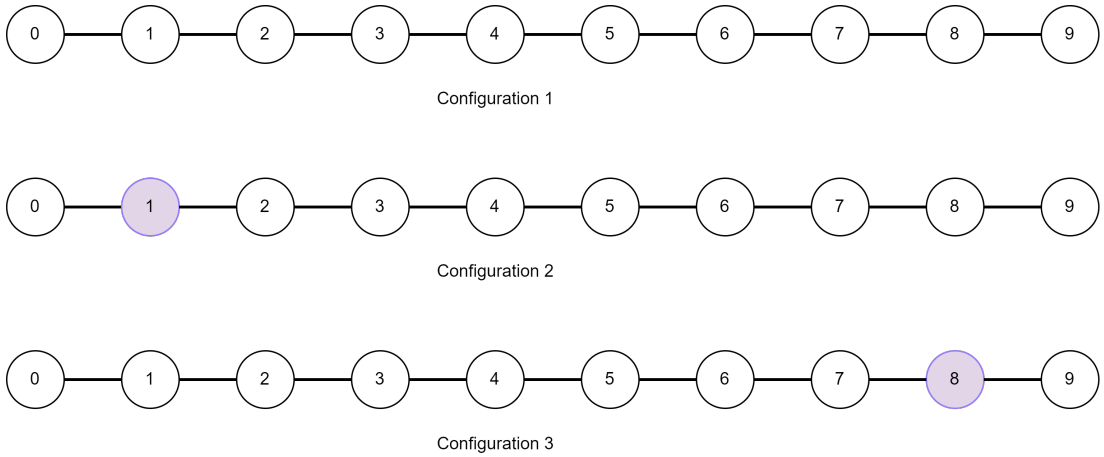


Figure 7.2: Three spanning branch configuration with configuration 2 and 3 with a lower clock quality time-aware system (in purple)

Node	1	2	3	4	5	6	7	8	9
δr_i ($\times 10^{-7}$))	0.497	0.994	1.49	1.99	2.49	2.98	3.48	3.98	4.47
δC_i (ns)	**102.38**	**204.8**	**267.26**	**329.78**	**392.34**	**454.96**	**517.63**	**580.35**	**643.12**
δGM_i (ns)	**102.33**	**204.70**	**267.11**	**329.57**	**392.09**	**454.65**	**517.27**	**579.94**	**642.65**
P_i^U (µs)	**7.602**	**2.705**	**2.767**	**2.830**	**2.892**	**2.955**	**3.017**	**3.080**	**3.143**

Table 7.3: δr_i, δC_i, δGM_i and P_i^U of the nodes of configuration 2. Values in bold are superior to the ones of configuration 1.

Node	1	2	3	4	5	6	7	8	9
δr_i ($\times 10^{-7}$))	0.497	0.994	1.49	1.99	2.49	2.98	3.48	3.98	4.47
δC_i (ns)	62.36	124.76	187.22	249.73	312.29	374.9	437.57	**540.31**	**643.1**
δGM_i (ns)	62.31	124.67	187.07	249.53	312.04	374.6	437.21	**540**	**562.59**
P_i^U (µs)	2.562	2.625	2.687	2.750	2.812	2.875	2.937	**8.04**	**3.143**

Table 7.4: δr_i, δC_i, δGM_i and P_i^U of the nodes of configuration 3. Values in bold are superior to the ones of configuration 1.

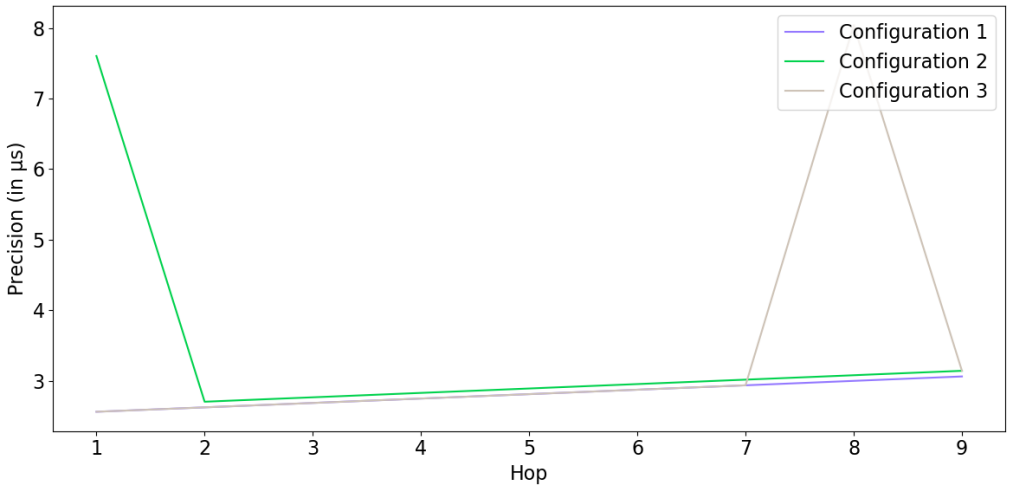


Figure 7.3: Upper bound on the precision of the three configurations as a function of the number of hops

configuration 3 between nodes 2 and 7 and between configuration 1 and the other two configurations at node 9. We also observe that regardless of the placement of the lower quality node, the precision of node 9 is the same for configuration 2 and 3 but remains higher than the one of configuration 1 (without the lower quality node). However, in the case of configuration 2, more nodes suffer from the lower precision.

As assumed, a lower quality time-aware system near the root impacts the precision of the nodes that are later in the spanning tree. However, the impact in our example is only 80ns, which remains very low compared to the precision limit which is between 2.5µs and 3.1µs for the good quality node. In addition, in a critical embedded network, if savings must be made, they will not be made on the quality of the clocks of the switches but rather of the end-stations which are at the end of the spanning tree. Indeed, these switches remain a central element in the synchronisation of the network and in the distribution of useful messages that may require the use of the Time-Aware Shaper and therefore a need for precise synchronization. Thus, we conclude that the position of lower quality clocks is not worth optimizing for critical embedded networks.

In view of the previous results, we will only use the $M_{Dist}(x)$ metric to evaluate the precision in the following.

7.1.2 A robust configuration

For robustness, our goal is to find combinations of several spanning trees that allow to contact from the root as many time-aware systems as possible in case of any link and/or node failure. Thus, in case of failures, the maximum number of time-aware systems remains synchronized. Therefore in this subsection, we begin by reviewing the literature to find an alternative to bruteforce with metric methods, then we present four metrics that we will test to study their discriminating power in order to select one.

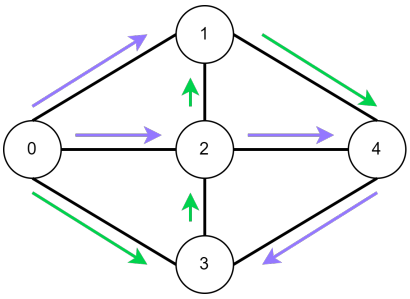


Figure 7.4: Illustration of two disjoint spanning trees rooted in node 0

Background The definition of robustness varies greatly depending on the network application field. In the following, we define this term and review the literature to see how to evaluate it.

In the IoT world, a robust spanning tree is a spanning tree where a node failure disconnects the least number of devices. But this means connecting as many nodes as possible to the central device, opening the door to a compromise between power consumption and robustness to failures, as explored by England et al. in [54].

In wired networks such as those forming the Internet, robustness to failure is based on dynamic reconfiguration of the routes to be taken, using generally distributed algorithms such as Spanning Tree Protocol (STP)[8] or its evolutions such as Rapid Spanning Tree Protocol (RSTP)[14] or Multiple Spanning Tree Protocol (MSTP)[13] for level 2 of the OSI model. In this way, when a device is added or lost, a new spanning tree is created for data transmission on the network. BMCA is a similar algorithm that uses the same principle to create the spanning tree that will be used to distribute synchronization information.

In critical networks such as AFDX networks, robustness is also very important, due to the much higher level of criticality of the data passing through the network. Indeed, the loss of a message from a DAL-A system could have catastrophic consequences. To reduce the probability of such an event occurring in avionics networks, there are in fact two identical AFDX networks running in parallel. End stations transmits their messages on both networks via two ports assigned respectively to each network. The message will then follow the same route on both networks and will be received by both ports of the receiving end station. In TSN, we find a similar mechanism called Frame Replication and Elimination for Reliability (FRER)[30], but not based on the use of two networks. In fact, this mechanism simply enables Ethernet frames to be replicated, so that several routes are transported independently on the same network. So, in critical networks, we have static route duplication, to avoid the reconfiguration times of dynamic mechanisms, but not full spanning tree duplication.

In our case, we do not have a path or a single spanning tree that must offer robustness, but a set of spanning trees whose cardinality is the number of domain. In the literature, we find works dealing with the robustness of a set of spanning trees in the field of graph theory. The first interesting concept is the disjoint (or edge-disjoint) spanning tree. A set of spanning trees is said to be disjoint if none of these spanning trees has a link in common with any other spanning tree in the set. A set of two disjoint spanning trees is illustrated on Figure 7.4. The literature contains numerous works to find such spanning tree sets for particular graphs, such as star [59] or twisted cubes (hypercube variants)[65]. In our case, this type of set is very robust to a link failure. Indeed, $k-1$ link failures have no impact on the distribution of `SYNC` messages if we have k disjoint spanning trees. However, one or more node failures, (for example node 2 in Figure 7.4) can have a significant impact on the message distribution.

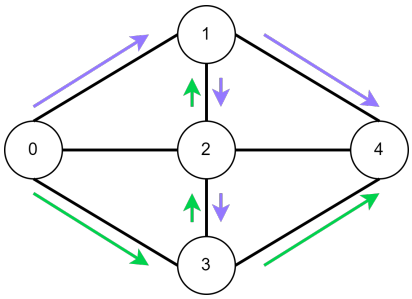


Figure 7.5: Illustration of two independent spanning trees rooted in node 0

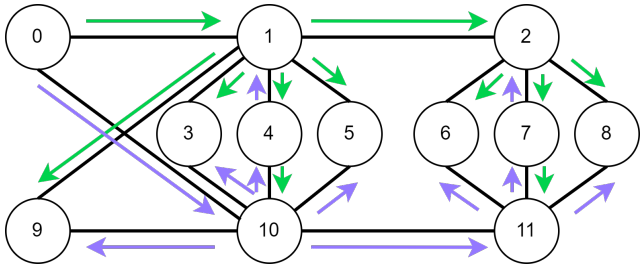


Figure 7.6: Two independent spanning trees rooted in node 0 on the satellite topology

Such weak points of failure can be eliminated with the independent spanning trees concept. A set of spanning trees is said to be independent if, for any vertex v of the graph, all paths between the root and v are vertex disjoint. A set of two independent spanning trees is depicted in the Figure 7.5. As for the disjoint spanning tree sets, many works have been done to propose algorithms to enumerate them or to find them but being applicable only to limited graph families such as hypercubes [86]. Some of these families of graphs are usable in real situations, such as processor interconnection using hypercubes in multiprocessor architectures like the Intel iPSC. In the case of embedded networks, it is very difficult to obtain the independence of spanning trees, but real applications remain for the moment limited. Some networks considered as very critical offer two independent spanning trees like the satellite network, as depicted in Figure 7.6, due to its construction inherited from the double redundant MIL-STD-1553 bus or like the double AFDX network. However, one of the interests of TSN is the mixed criticality. We can therefore imagine a very connected critical network, for example the ventilation system in the cabin of an airplane, to which are added non-critical devices connected by a single link, such as buttons to call the staff.On such network, these unique links prevent the existence of independent or even disjoint spanning trees, like for the automotive network recalled in Figure 7.7. On this network, although two independent paths between some switches can be found like between switch 1 and switch 4, links between switch 0 and switch 5 or between any end-station and its access switch does not allow for an independent or even disjoint spanning tree set.

Robustness metrics definition Because of the heterogeneous topologies of our networks, finding a robust spanning tree set is not as simple as finding a disjoint or independent set. Therefore, we propose to define multiple metrics to compare the sets between them. Following metrics are investigated next:

- Metric 1 : Disjointedness

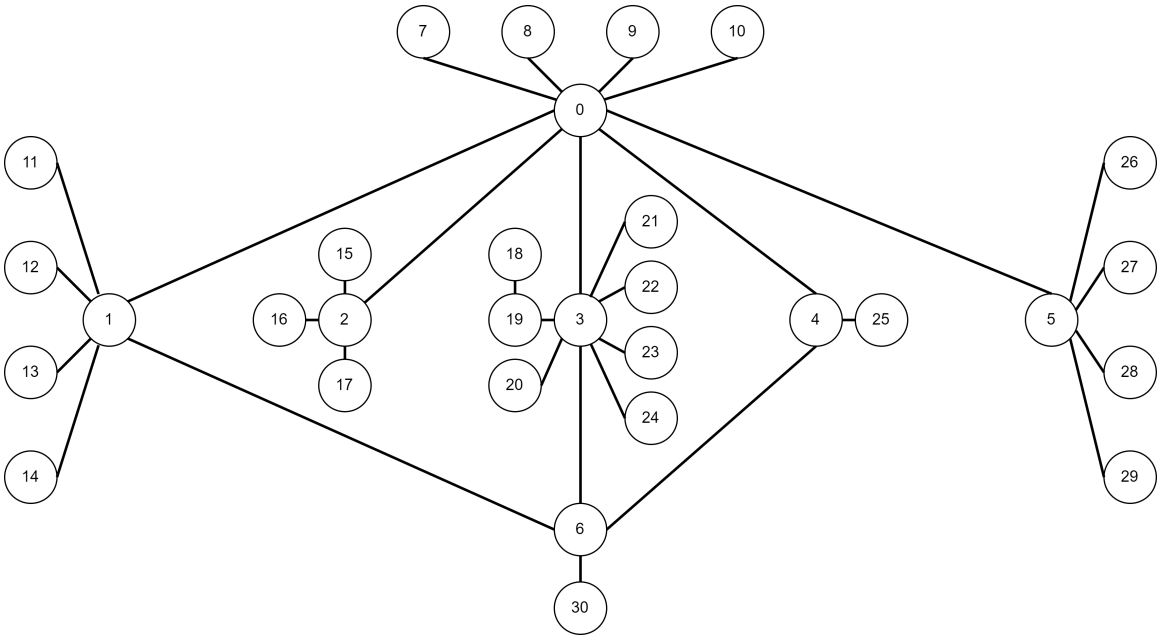


Figure 7.7: Automotive network graph

- Metric 2 : Independence
- Metric 3 : Resistance
- Metric 4 : Failure impact

First (respectively second) robustmess metric aims to quantify the disjointness (respectively independence) of a spanning tree set X. To do so, it measures the number of nodes accessible from the root using disjoint (respectively independent) path. Thus, by maximizing this metric, we find the spanning tree sets connecting the most nodes with disjoint or independent paths from the root. Formally, these two metrics are calculated with the following equations:

$$M_{Disjointedness}(X) = \sum_{i=0}^{n} importance_i * isPathDisjoint_i(X) \tag{7.2}$$

$$M_{Independence}(X) = \sum_{i=0}^{n} importance_i * isPathIndependent_i(X) \tag{7.3}$$

Where n is the number of time-aware systems in the graph, $importance_i$ is the importance factor of the time-aware system i, $isPathDisjoint_i$ returns 0 (respectively 1) if the path is not (respectively is) disjoint and $isPathIndependent_i$ returns 0 (respectively 1) if the path is not (respectively is) independent between the Grandmaster and the time-aware system i with the spanning tree set X.

The third robustness metric counts, for each node, the minimum number of failures needed to disconnect it from the Grandmaster when using spanning trees from the set under evaluation. This metric being intended to evaluate the robustness between several spanning trees with the same root, only the failures of links and nodes are taken into account. Thus, by maximizing this metric each node will be reachable by paths more robust to failure. Formally, it is expressed by the following

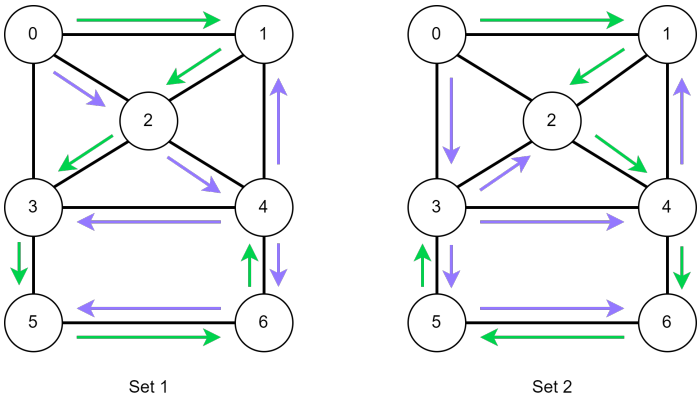


Figure 7.8: Two different spanning tree sets rooted in node 0 on AFDX A350 core topology.

equation:

$$M_{Resistance}(X) = \sum_{i=0}^{n} importance_i * minFailure_i(X) \tag{7.4}$$

Where n is the number of time-aware systems in the graph, $importance_i$ is the importance factor of the time-aware system i, $minFailure_i$ is a function that returns the minimum number of failures it takes to disconnect the time-aware system i from the spanning tree of the set X.

The last robustness metric takes the problem in the other direction by iterating on the failures instead of iterating on the nodes. Thus, for each possible failure, it sums the number of reachable nodes. Thus minimizing this metric selects the sets where failures impact the least number of nodes. The iteration takes place over all the combinations of k link failures, k node failures and k concurrent link and node failures. Here, k is the number of domains minus 1 because, at best, it takes n failures (except root failure) to disrupt the distribution of synchronization information with n domains. This metric is calculated using the following equation:

$$M_{FailureImpact}(X) = \sum_{j=0}^{m} \sum_{i=0}^{n} importance_i * isReachable_i(X, j) \tag{7.5}$$

Where m is the number of failures possible in the graph, n is the number of time-aware systems in the graph, $importance_i$ is the importance factor of the time-aware system i, $isReachable_i$ is a function that returns 0 (respectively 1) if the node i is not (respectively is) reachable with the spanning tree set X after failure j. By construction, this metric meets our need for robustness evaluation but its complexity to calculate makes it slower.

Robustness metrics evaluation Now that we have presented the four robustness metrics, our goal is to find the most discriminating metric, as with precision. Ideally, we are looking for a metric that would give results similar to $M_{FailureImpact}$ but simpler to compute. As previously, the results presented are a subset of the results that we studied to draw our conclusions.

Let's start with a study on two different sets of spanning trees originating from the A350 AFDX core topology as depicted in Figure 7.8. The scores obtained on these sets for each metric are presented in Table 7.5. The ranking from most to least robust for each metric of the two sets is summarized in Table 7.6.

	Set 1	Set 2
$M_{Disjointedness}$	6	6
$M_{Independence}$	2	6
$M_{Resistance}$	8	12
$M_{FailureImpact}$	109	113

Table 7.5: Score obtained by the two sets for the four metrics

Position	1	2
$M_{Disjointedness}$	Set 2 & Set 1	
$M_{Independence}$	Set 2	Set 1
$M_{Resistance}$	Set 2	Set 1
$M_{FailureImpact}$	Set 2	Set 1

Table 7.6: Ranking of the two sets by the four metrics

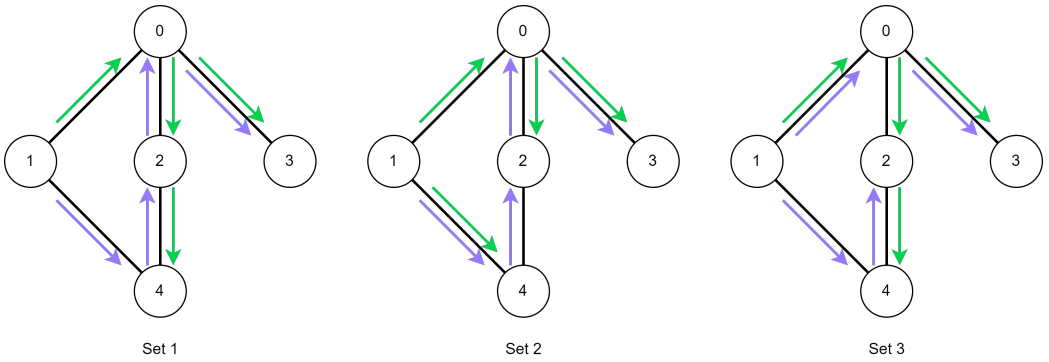


Figure 7.9: Three different spanning tree sets rooted in node 1 on a simplified automotive core topology.

On this topology, $M_{Disjointedness}$ gives a different ranking than the other metrics. Indeed, this metric can't make a difference between Set 1 and Set 2 whereas Set 1 is clearly less robust than Set 2 due to the central location node 2 takes in the distribution. This example underlines the limit of the metric 1. This metric can be useful to avoid sets that are weak to link failure but can't detect sets weak to node failure, while metrics 2,3 and 4 can. Thus, the discriminant power of metric 1 is too limited for our purposes.

In addition, we observe that the ranking of metric 2, 3 and 4 are identical for the two sets. This result is a limitation of the topology used for this illustration. Using a less connected topology like the automotive core topology, we can observe a difference in the ranking between these metrics. To do this, we use the three sets rooted in node 1 shown in Figure 7.9. To simplify this illustration, we use a simplified version of the automotive topology. The results of the metrics are presented in Tables 7.7 and 7.8.

In this example, all metrics produce the same result except for metric 4, $M_{FailureImpact}$, which

	Set 1	Set 2	Set 3
$M_{Disjointedness}$	3	2	2
$M_{Independence}$	3	2	2
$M_{Resistance}$	7	6	6
$M_{FailureImpact}$	39	38	37

Table 7.7: Score obtained by the three sets at the four metrics

Position	1	2	3
$M_{Disjointedness}$	Set 1		Set 2 & Set 3
$M_{Independence}$	Set 1		Set 2 & Set 3
$M_{Resistance}$	Set 1		Set 2 & Set 3
$M_{FailureImpact}$	Set 1	Set 2	Set 3

Table 7.8: Ranking of the three sets by the four metrics

produces a discriminating ranking, as shown in 7.8.

Let's start with sets where all metrics produce the same result, namely sets 1 and 3. The four metrics agree to elect Set 1 as the most robust of the three sets. Indeed, with this set, all paths that can be independent are independent. Thus, they are also disjoint and resistant to failure, and failures have low impact. The same reasoning is applicable for Set 3 which is elected as the least robust by the four metrics because of the paths between the root and node 0 and the root and node 3. Indeed, these two paths are identical in the two spanning trees of the set and therefore not robust to failure, neither disjoint nor independent.

The difference between the metrics takes place for Set 2 which is considered last by the first three metrics and second according to $M_{FailureImpact}$. This difference is caused by the path between the node 1(Grandmaster) and node 3. This path is considered by the first three metrics as non-disjoint, non-independent and thus requires a single failure to be disconnected due to the link (0,3). This binary behaviour prevents the search for robustness at the beginning of the path. Thus, the first three metrics can't see the difference between Set 2 and Set 3. While metric 4, by assessing the impact of a failure, measures a difference between the two sets. Indeed, the latter produces a different result for sets 2 and 3 on this specific path because it is not limited only to the weak link or node but takes into account the whole path. Metric 4 takes into account that the paths used up to node 0 are redundant to a link or node failure in the case of Set 2 and not in the case of Set 3. This thus leads to a power discriminating more important for metric 4 because it is able to study all the paths and not just the weakest link.

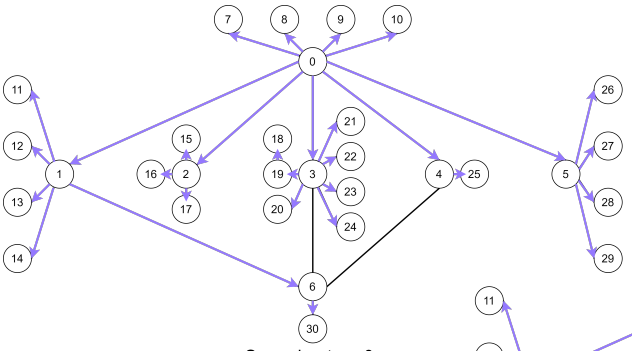
This limitation of the first three metrics illustrated here on a simplified topology will occur for each end-station of the automotive case study, connected by a single link (like node 3) but also in a network where the connectivity varies like the AFDX network.

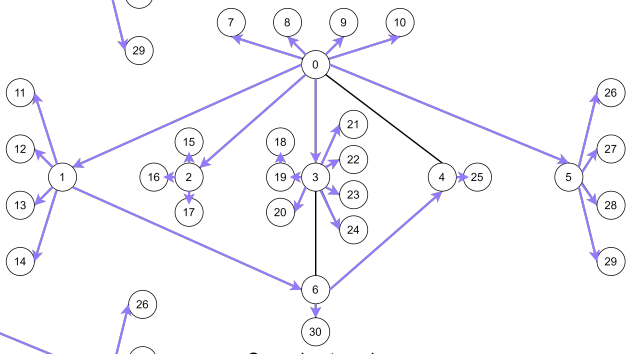
Selection These observations are generalizable to all the cases study in the EDEN project. The three first metrics can't provide the same result as $M_{FailureImpact}$ with their simpler evaluation method. Thus, we will use the metric $M_{FailureImpact}$ for its discriminating power previously shown, despite its greater complexity, to evaluate the robustness of a set of spanning trees in the following.

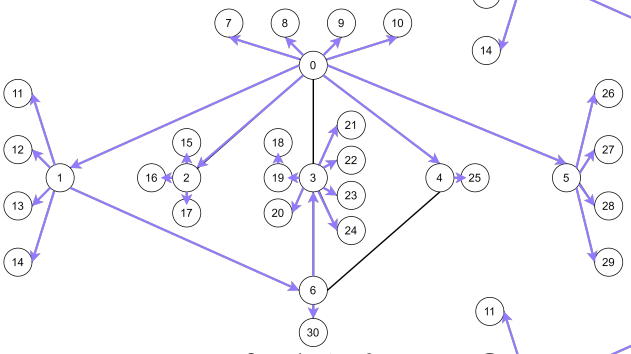
7.2 Relation between robustness and precision

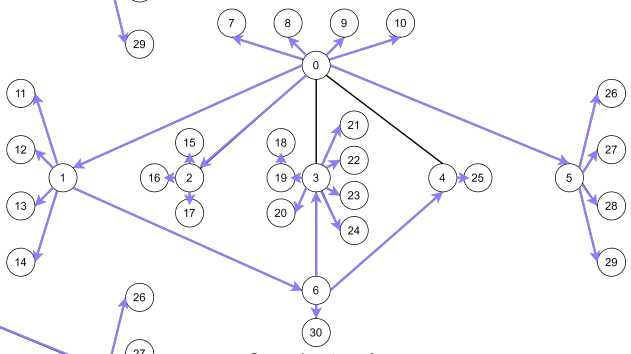
Now that we have two metrics to evaluate precision and robustness, let's study the relationship between these two parameters. To realize this study, we will use the automotive network because it corresponds well to the concept of mixed criticality that are expected for the use of TSN. In order to best illustrate the relationships between both metrics, we assume that the only Grandmaster is located on node 0. For the moment, we are only interested in link and node failures. We will start with two domains (i.e. two spanning trees) and then study three-domain configurations in order to cover the three independent paths between node 0 and node 6.

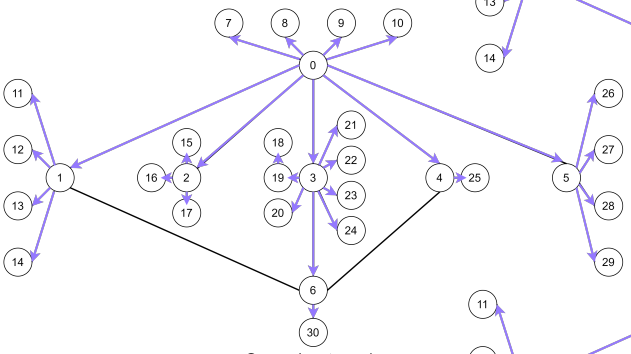
On this topology 12 spanning trees are possible as illustrated in Fig 7.10. Thus, 66 combinations of two domains (2 out of 12) and 220 of three domains (3 out of 12) are possible on this topology. Using our two metrics should help us reduce this set of possibilities. Thus, the different score and ranking are presented in Table 7.9 for the precision and Table 7.10 and Table 7.11 for the robustness.

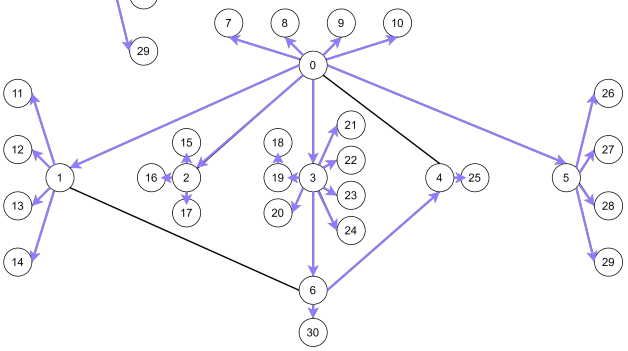

Spanning tree 0


Spanning tree 1


Spanning tree 2


Spanning tree 3


Spanning tree 4


Spanning tree 5

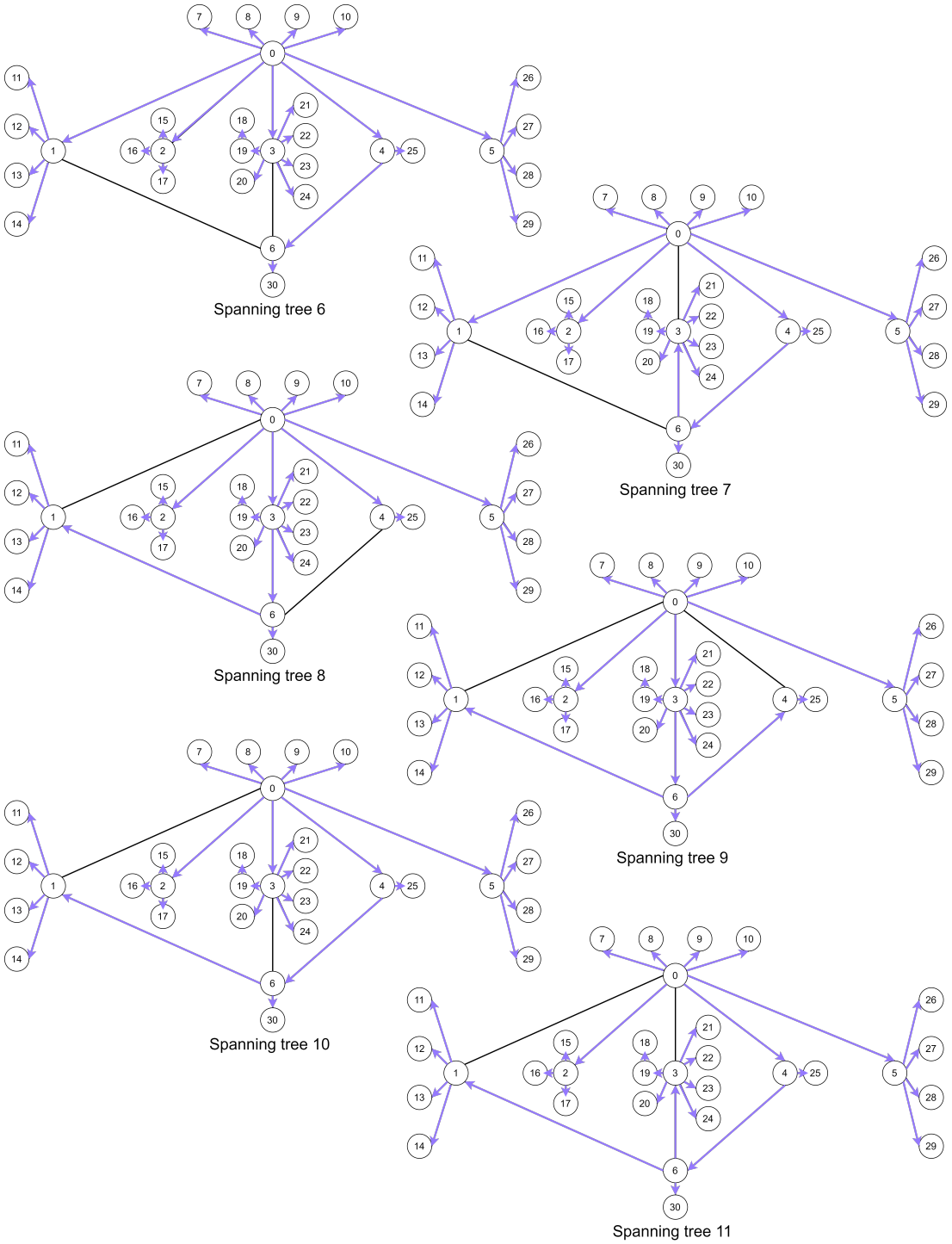


Figure 7.10: The 12 spanning trees of the automotive network graph rooted in node 0

Spanning tree	Precision score	Rank
0	**53**	**1**
1	57	2
2	69	5
3	73	6
4	**53**	**1**
5	57	2
6	**53**	**1**
7	69	5
8	63	3
9	67	4
10	63	3
11	79	7

Table 7.9: Score and ranking obtained for the 12 spanning trees rooted in 0 of the automotive topology with the precision metric.

First, if we focus on the results of the precision metric, we observe as expected that spanning trees using the least number of hops on average such as spanning tree 0 are better ranked than spanning trees not minimizing the number of hops between the time-aware systems and the root like spanning tree 11. Indeed, with this spanning tree, a `Sync` message leaving the root to reach the node 1 will then carry out the path $0 \rightarrow 4 \rightarrow 6 \rightarrow 1$ whereas with the spanning tree 0 this same transmission is direct.

On the other hand, this spanning tree 11 obtains a very good ranking in the robustness metric when it is coupled with spanning tree 1 (rank: first) or with spanning tree 5 (rank: first). The same is true for the three-domain sets, where it is found in the sets with the first three best ranks. This good score is explained by the fact that this spanning tree uses a long path to contact switch 1 (and all the end-stations connected to it) and a short path to contact switch 4 (and all the end-stations connected to it) at the opposite of the spanning tree 1 and 5. This combination of short/long path avoids using the same path or very similar paths to contact the same time-aware systems, and thus offers paths that are more resistant to failure.

Spanning tree set	Robustness score	Rank
(0, 1)	134	17
(0, 2)	128	11
(0, 3)	126	9
(0, 4)	130	13
(0, 5)	128	11
(0, 6)	130	13
(0, 7)	122	5
(0, 8)	125	8
(0, 9)	123	6
(0, 10)	125	8
(0, 11)	117	2
(1, 2)	126	9
(1, 3)	136	19

Spanning tree set	Robustness score	Rank
(1, 4)	128	11
(1, 5)	132	15
(1, 6)	128	11
(1, 7)	120	4
(1, 8)	123	6
(1, 9)	127	10
(1, 10)	123	6
(1, 11)	**115**	**1**
(2, 3)	166	25
(2, 4)	122	5
(2, 5)	120	4
(2, 6)	122	5
(2, 7)	138	20
(2, 8)	117	2
(2, 9)	**115**	**1**
(2, 10)	117	2
(2, 11)	133	16
(3, 4)	120	4
(3, 5)	124	7
(3, 6)	120	4
(3, 7)	136	19
(3, 8)	**115**	**1**
(3, 9)	119	3
(3, 10)	**115**	**1**
(3, 11)	131	14
(4, 5)	134	17
(4, 6)	130	13
(4, 7)	122	5
(4, 8)	131	14
(4, 9)	129	12
(4, 10)	125	8
(4, 11)	117	2
(5, 6)	128	11
(5, 7)	120	4
(5, 8)	129	12
(5, 9)	139	21
(5, 10)	123	6
(5, 11)	**115**	**1**
(6, 7)	128	11
(6, 8)	125	8
(6, 9)	123	6
(6, 10)	131	14
(6, 11)	123	6
(7, 8)	117	2
(7, 9)	**115**	**1**

Spanning tree set	Robustness score	Rank
(7, 10)	123	6
(7, 11)	163	24
(8, 9)	154	23
(8, 10)	135	18
(8, 11)	127	10
(9, 10)	133	16
(9, 11)	125	8
(10, 11)	148	22

Table 7.10: Score and ranking obtained by the 66 2-spanning tree sets rooted in 0 for the automotive topology with the robustness metric.

Spanning tree set	Robustness score	Rank
(0, 1, 11)	7159	11
(0, 2, 11)	7245	15
(0, 3, 11)	7135	7
(0, 4, 11)	7251	16
(0, 5, 11)	7141	8
(0, 6, 11)	7269	18
(0, 7, 11)	7269	18
(0, 8, 11)	7236	14
(0, 9, 11)	7126	6
(0, 10, 11)	7269	18
(1, 2, 11)	7135	7
(1, 3, 11)	7135	7
(1, 4, 11)	7141	8
(1, 5, 11)	7135	7
(1, 6, 11)	7159	11
(1, 7, 11)	7159	11
(1, 8, 11)	7126	6
(1, 9, 11)	7120	5
(1, 10, 11)	7159	11
(2, 3, 11)	8127	53
(2, 4, 11)	7227	13
(2, 5, 11)	7117	4
(2, 6, 11)	7245	15
(2, 7, 11)	8237	55
(2, 8, 11)	7212	12
(2, 9, 11)	7102	2
(2, 10, 11)	7245	15
(3, 4, 11)	7117	4
(3, 5, 11)	7111	3
(3, 6, 11)	7135	7
(3, 7, 11)	8127	53

Spanning tree set	Robustness score	Rank
(3, 8, 11)	7102	2
(3, 9, 11)	**7096**	**1**
(3, 10, 11)	7135	7
(4, 5, 11)	7159	11
(4, 6, 11)	7269	18
(4, 7, 11)	7269	18
(4, 8, 11)	7254	17
(4, 9, 11)	7144	9
(4, 10, 11)	7269	18
(5, 6, 11)	7159	11
(5, 7, 11)	7159	11
(5, 8, 11)	7144	9
(5, 9, 11)	7144	9
(5, 10, 11)	7159	11
(6, 7, 11)	7605	37
(6, 8, 11)	7254	17
(6, 9, 11)	7144	9
(6, 10, 11)	7605	37
(7, 8, 11)	7254	17
(7, 9, 11)	7144	9
(7, 10, 11)	7605	37
(8, 9, 11)	7768	44
(8, 10, 11)	7878	50
(9, 10, 11)	7768	44

Table 7.11: Score and ranking obtained by the 54 3-spanning tree sets including spanning tree 11 and rooted in 0 for the automotive topology with the robustness metric. The full Score and ranking obtained by the 220 3-spanning tree set rooted in 0 of the automotive topology with the robustness metric can be found in Appendix C.

Results are summarized in Figure 7.11 which displays, for each 2-spanning tree sets, the precision score as a function of the robustness score. We observe that a robust set (lower robustness score) does not offer the best precision (lower precision score) as illustrated previously with the spanning tree 11. There are 4 points that are Pareto-optimal, i.e. that dominate all other points and that don't dominate each other. A point dominates another one if its two metric values are better (lower in this case). Each Pareto-optimal points on this figure corresponds to one or more 2-spanning tree sets. To identify them, we highlight them in green in Figure 7.11 and in Table 7.10. From these observations, we can conclude that there is not a unique Pareto-optimal but in our example three for the robustness and one for the precision. As the precision is configurable by means other than the number of hops (quality of clocks, *syncInterval*, ...), we will first seek to maximize robustness. However, as we can see in the example, the three robustness Pareto-optimal can be separated thanks to their score obtained from the precision metric. Thus and for the continuation, we define our optimal configuration as the most precise configuration among the most robust ones.

It's also worth noting that not all sets containing spanning tree 11 rank equally well. Take the example of set (8,10,11) in Table 7.11, which is near the bottom of the ranking, in 50th place out of 56. This can be explained by the strong similarity of these three spanning trees, preventing the

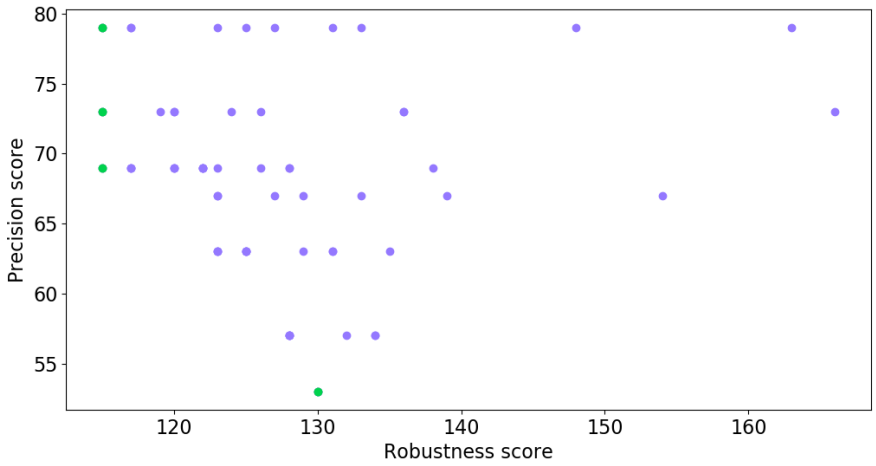


Figure 7.11: Precision score according to the robustness score for the 66 sets of two domains rooted in 0 in the automotive topology

existence of partially independent paths.

7.3 Synergies between multiple Grandmasters domains

Until now, we have been interested in link or node failures disrupting the distribution of `Sync` and `Follow_Up` by breaking spanning trees. However, the root of the spanning tree (i.e. Grandmaster) might also fail and such a failure is critical, as this root is a central element of the synchronization distribution. Thus, to withstand a Grandmaster failure with static configuration, it is necessary to use several Grandmasters. In this section, we propose to optimise the choice of spanning tree sets for multiple Grandmasters in order to create combinations of sets that synergise to increase robustness.

As a reminder, there are two types of Grandmasters: ”classic” and hot standby Grandmasters. The difference is that the hot standby Grandmaster synchronizes on one of the ”classic” Grandmasters and distributes this time to other domains while a ”classic” Grandmaster will simply use the time from its time source (Internal oscillator, GPS, ...) to distribute it on its domains.

The robustness evaluation method presented above can also be applied to a several Grandmasters configuration. Let’s take the previous example with two domains from the automotive case study with node 0 as GM. To avoid being impacted by the loss of one GM, let’s add a second GM, node 4, which also has two domains. In this case, it is possible to apply the robustness evaluation by iterating over the set of unique possible failures. Such an evaluation would find the most robust configuration to a single failure. However, a 4-domain configuration offers increased robustness, in some cases withstanding up to 3 failures (number of domains - 1) due to domain synergies. It is therefore possible to perform an evaluation by iterating over all combinations of 3 or fewer failures. Inevitably, this increases the number of failures to be studied, leading to an increase in the time required for evaluation. In the following, sequential evaluation will rather be use. Indeed, a selection of the most robust configurations for each GM is made first. In our example, it is a 1 failure robustness evaluation for the two GM. Next, the 4-domain robustness evaluation can be performed only on the previously selected domains, thus reducing the number of combination of two

Spanning tree set	Robustness score	Robustness rank	Precision score	Precision rank
(0, 3)	127	1/34	104	5/7
(0, 9)	127	1/34	104	5/7
(3, 4)	127	1/34	104	5/7
(4, 9)	127	1/34	104	5/7
[...]	[...]	[...]	[...]	[...]
(5, 9)	233	34/34	114	6/7

Table 7.12: Selection of the robustness and precision score and rank for the 2-spanning tree sets rooted in 4 on the automotive topology. The precision score display is the worst of the two spanning trees of a set

2-spanning trees and consequently reducing the evaluation time. In our example, studying only the synergies of the most robust sets limits the search space from 4356 (66^2) to 24 combination of set.

To illustrate the study of the inter-Grandmaster synergies, we base ourselves on the results presented previously in the context of the 2-domain study of the automotive configuration having for root node 0, i.e. Table 7.9 and Table 7.10. A sub-set selected to illustrate our remarks, of the evaluation of the precision and the robustness of the sets of spanning trees with node 4 as root is presented in Table 7.12. These two rankings allow us to select the most robust set of 2 spanning trees for each Grandmaster, i.e. (1, 11), (2, 9), (3, 8), (3, 10), (5, 11) and (7, 9) for node 0 and (0, 3), (0, 9), (3, 4) and (4, 9) for node 4. The 24 (6×4) possible combinations of the two sets are then studied with the robustness metric. The results are described in Table 7.13.

First, we observe a 1000 times higher robustness score when studying the four domains together compared to when studying only the two domains of a single Grandmaster. This is due to the much larger number of failures studied. Indeed, when the metric is computed for two domains, the iteration takes place on all the unique failures possible against the combination of all the unique failures, two failures and three failures when the metric is computed for four domains. Thus, in our example, 63 failure combinations are studied for two domains against 41727 failure combinations for four domains, naturally increasing the score. Secondly, there is a difference between the scores for the different combinations. If we focus on the combination of sets (2,9) and (0,3) ranked first and the combination of sets (1,11) and (0,3) ranked last, we observe that a failure of the links (0,1) and (4,6) doesn't have the same impact on the two configurations as illustrated in Fig 7.12. Indeed, with the first configuration the distribution of messages is not impacted by the two link failures, while with the second configuration the `Sync` and `Follow_Up` messages coming from the two Grandmasters are no longer received by the nodes 1 and 6. Thus, this additional resistance to some combinations of failure discriminates some combinations of sets, allowing us to identify sets which, when combined, have a good synergy thus increasing robustness to failures.

However, it should be noted, that this study of synergies between domains of different Grandmasters allows for an over-dimensioned resistance to failure compared to the initial need which is one Grandmaster's failure resistance. For example, in the automotive case study, the need is the resistance to one failure on the robust part of the network but thanks to this study we find few configuration that can resist to some combination of two failures. This study can therefore be optional for the network designer if he considers that it is not necessary to look for the most robust configuration possible with the different domains he has at his disposal.

Spanning tree sets	Robustness score	Robustness rank
(2, 9) and (0, 3)	188131	1/7
(2, 9) and (0, 9)	188131	1/7
(2, 9) and (3, 4)	188131	1/7
(2, 9) and (4, 9)	188131	1/7
(3, 8) and (0, 3)	188131	1/7
(3, 8) and (0, 9)	188131	1/7
(3, 8) and (3, 4)	188131	1/7
(3, 8) and (4, 9)	188131	1/7
(3, 10) and (3, 4)	188996	2/7
(3, 10) and (4, 9)	188996	2/7
(7, 9) and (0, 3)	189307	3/7
(7, 9) and (0, 9)	189307	3/7
(3, 10) and (0, 3)	189314	4/7
(3, 10) and (0, 9)	189314	4/7
(7, 9) and (3, 4)	189625	5/7
(7, 9) and (4, 9)	189625	5/7
(1, 11) and (3, 4)	189836	6/7
(1, 11) and (4, 9)	189836	6/7
(5, 11) and (0, 3)	189836	6/7
(5, 11) and (0, 9)	189836	6/7
(1, 11) and (0, 3)	190154	7/7
(1, 11) and (0, 9)	190154	7/7
(5, 11) and (3, 4)	190154	7/7
(5, 11) and (4, 9)	190154	7/7

Table 7.13: Robustness score and rank according to the combination of the most robust set from Grandmaster 0 and 4 in the automotive topology

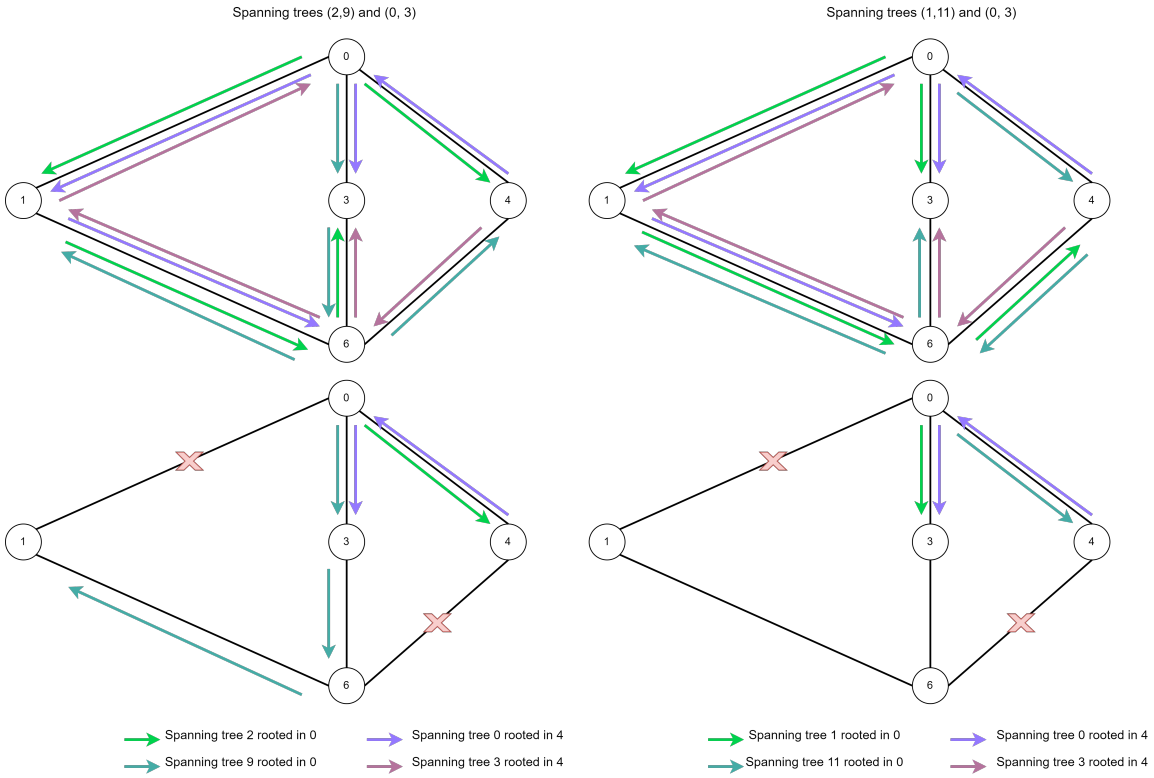


Figure 7.12: Representation of (2,9)/(0,3) and (1,11)/(0,3) spanning tree set combinations without failure (top line) and with failure (bottom line) on links (0,1) and (4,6). A simplified version of the topology, without node connected by one link, is represented here to help the reading. The (1,11)/(0,3) combination, upon failures, is disconnecting nodes 6 and 1 from the Grandmaster 0.

7.4 Multiple time base issue

Another aspect of static use, when using hot standby Grandmasters, is the possibility of two different time bases appearing in the same network in case of failures . In this section, we propose several ways of avoiding this problem and select one for further use.

7.4.1 Problem statement

This problem will be called in the following multiple time base issue. Indeed, as illustrated in Figure 7.13, depending on their locations, failures can lead to a disconnection of the Grandmaster in hot standby (node 4) form the classical Grandmaster (node 0). Moreover, because of these failures, a part of the network can synchronize on the main Grandmaster (node 1 and 2 in the example) while the rest of the network synchronizes on the Grandmaster in hot standby (node 3 in the example). As the latter does not receive any more synchronization message from the main Grandmaster, its clock drifts, creating a second time base in the same network.

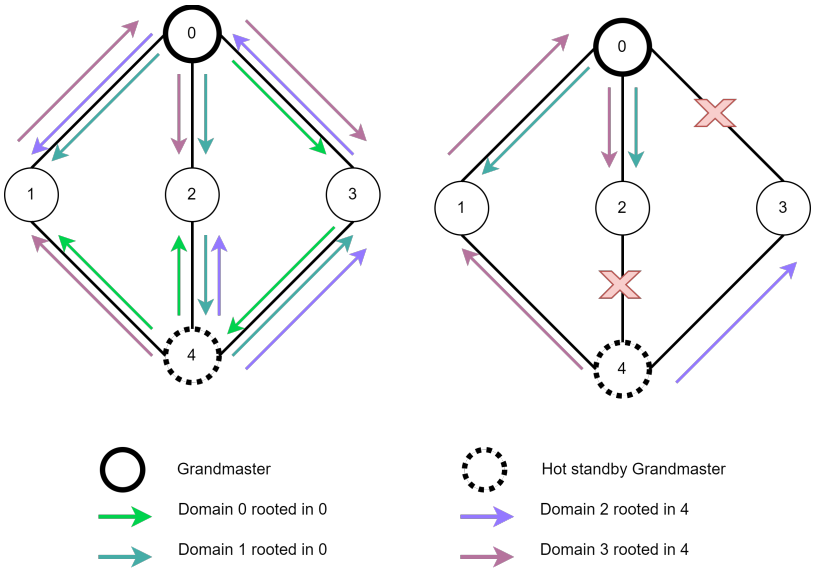


Figure 7.13: Illustration of the multiple time bases issue with 2 Grandmasters and 4 domains.

These two time bases then have consequences on the travel time of the messages when using the TAS. For example, a message connecting node 3 to node 0 through node 4 and node 1 is first received by switch 4 which is in the hot standby Grandmaster time base. Once the time has come to open the TAS gate of the FIFO of this message, switch 4 sends the message to switch 1 where it is received at a possibly quite different moment in the TAS cycle of switch 1 because switch 1 follows another time base. The message must therefore wait for a possible later opening of its TAS gate. Thus, the message traversal time can be much greater than estimated during the network calculus study and therefore no longer meets the system constraints.

Furthermore, these two time bases can also impact distributed applications. Let's take for example the turning on of a car's lights. When the light up command is sent by the central ECU, the different ECUs receive the command and execute it in a synchronized way so that the powering of the different lights is in phase. So if the ECUs of the lights are not on the same time base,

the lights could turn on in an possibly completely out of phase way. In this section, we propose different ways of avoiding this issue. One of the proposed solutions is studied in details in order to be integrated with the other propositions of this chapter.

7.4.2 Proposed solutions

This multiple time base issue is only possible under the following two conditions:

1. The Grandmaster on hot standby no longer receives a synchronization message from the primary Grandmaster and drifts away.
2. A time-aware system is disconnected from the main Grandmaster but continues to synchronize on the hot standby Grandmaster.

We have to mitigate the occurence of these two conditions to avoid the multiple time base problem. To do this, we propose four solutions that prevent the appearance of the first condition.

The first solution is to use only classic Grandmasters (no Grandmaster in hot standby). For example, a network could have two devices able to synchronize on GPS time. The counterpart of this solution is that it is necessary to ensure that both devices always have the same time base and even in case of failure. The point is then to ensure each Grandmaster is connected to a GPS clock system.

The second solution is to use one or more dedicated PPS links to synchronize the Grandmaster in hot standby from the main Grandmaster. Such a Grandmaster in hot standby would then broadcast to its domains the time base coming from these dedicated links instead of the main Grandmaster time base coming from the network. However, this solution defeats the main purpose of the network, which is to share communication links between different uses.

The third solution to avoid the appearance of a second time base is to ensure that the drift that takes place between the two Grandmasters remains less than what is required by the specification of the different systems. Thus, by over-dimensioning the oscillator used by the Grandmaster in hot standby, it is possible to limit the drift in case of failure. Let's take for example the case of a digital audio study intended for the cabin of an aircraft. For this application, the synchronization precision requirement of the speakers is in the order of one millisecond. Let's suppose that the plane equipped with this system must land at least once every 24 hours (the A350's record is 14 hours). A 0.01 ppm crystal (the order of magnitude reachable with OCXO crystals) would lead to a maximum drift of 0.864ms in 24 hours and thus respect the 1ms constraint. However, the quality of the crystal capable of meeting this need in the case of a more constrained system is becoming less and less affordable. Indeed, let's assume that a TSN network replaces the AFDX network of such a plane and must ensure a precision under the microsecond. An oscillator with a drift below 10ppb is then necessary for the Grandmaster in hot standby. Such an oscillator would be a rubidium atomic clock. Moreover, it should be noted that in this case, the TAS and the applications must be sized to handle such a drift, which causes a significant over-dimensioning of the system.

The last solution is to make sure that it is impossible to disconnect the Grandmaster in hot standby from the main Grandmaster without splitting the network in two. To check if this issue is possible on the current configuration, it is necessary to enumerate all the paths having the main Grandmaster as source and the hot standby as a destination. If there are paths that don't belong to the domains synchronized by the main Grandmaster, multiple time bases may happen. To avoid this situation, we propose to add a new synchronization domain per uncovered path that ensures the hot standby Grandmaster is always receiving the main Grandmaster clock. It is necessary to create a domain per missing path because a node can't have multiple gPTP Slave port in the same domain from multiple links.

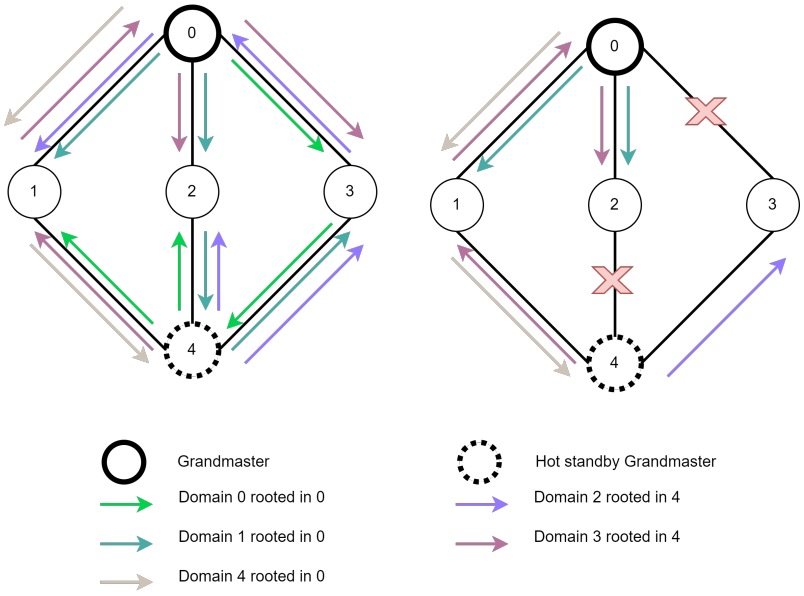


Figure 7.14: Illustration of a configuration that can't encounter the multiple time bases issue with 2 Grandmasters and 5 domains. Domain 4 is added to ensure all paths from Grandmaster 0 to hot standby Grandmaster 4 are covered with a domain rooted in 0.

Thus, in the previous example illustrated in Figure 7.13, it is enough to add a new domain that covers the previously uncovered paths connecting Grandmaster 0 with the hot standby Grandmaster 4, as shown in Figure 7.14. This solves this multiple time base issue. In this example, despite the failures, domain 4 keeps the synchronization between the Grandmaster in hot standby and the main Grandmaster in order to avoid the appearance of the second time base. Any combination of two failures cannot separate the two Grandmasters because the three paths from 0 to 4 are covered by a tree rooted in 0. Some combinations of three or more failures (such as failures of the links between 1,2,3 and 0), can cut all the links between the GMs but at the cost of splitting the network in two. It should be noted that the domain added in this example is not a spanning tree, but only covers the path not covered by the spanning trees of domains 0 and 1. Replacing domain 4 by a spanning tree is also possible at the cost of a slightly higher bandwidth consumption on the previously not impacted links. This solution is particularly interesting compared to the others proposed above. Its only disadvantage is an increase in the number of domains whose bandwidth cost has been shown to be very low earlier in this manuscript. We'll be using this solution next.

7.4.3 Multiple hot standby GMs

With several Grandmasters in hot standby, the previous reasoning is still applicable but faces a limit. To illustrate this, let's take the topology shown in Figure 7.15 as an example.

In this topology, node 0 is the main Grandmaster which synchronizes nodes 1 and 2 which are two Grandmasters in hot standby. The main Grandmaster 0 uses 3 domains to cover all possible paths between the different hot standby grandmasters, as described above. However, in case of a failure of the main Grandmaster, as illustrated in Fig 7.15-bottom left, the two Grandmasters in hot standby lose their reference and drift away independently. In this case, if some well-placed failures

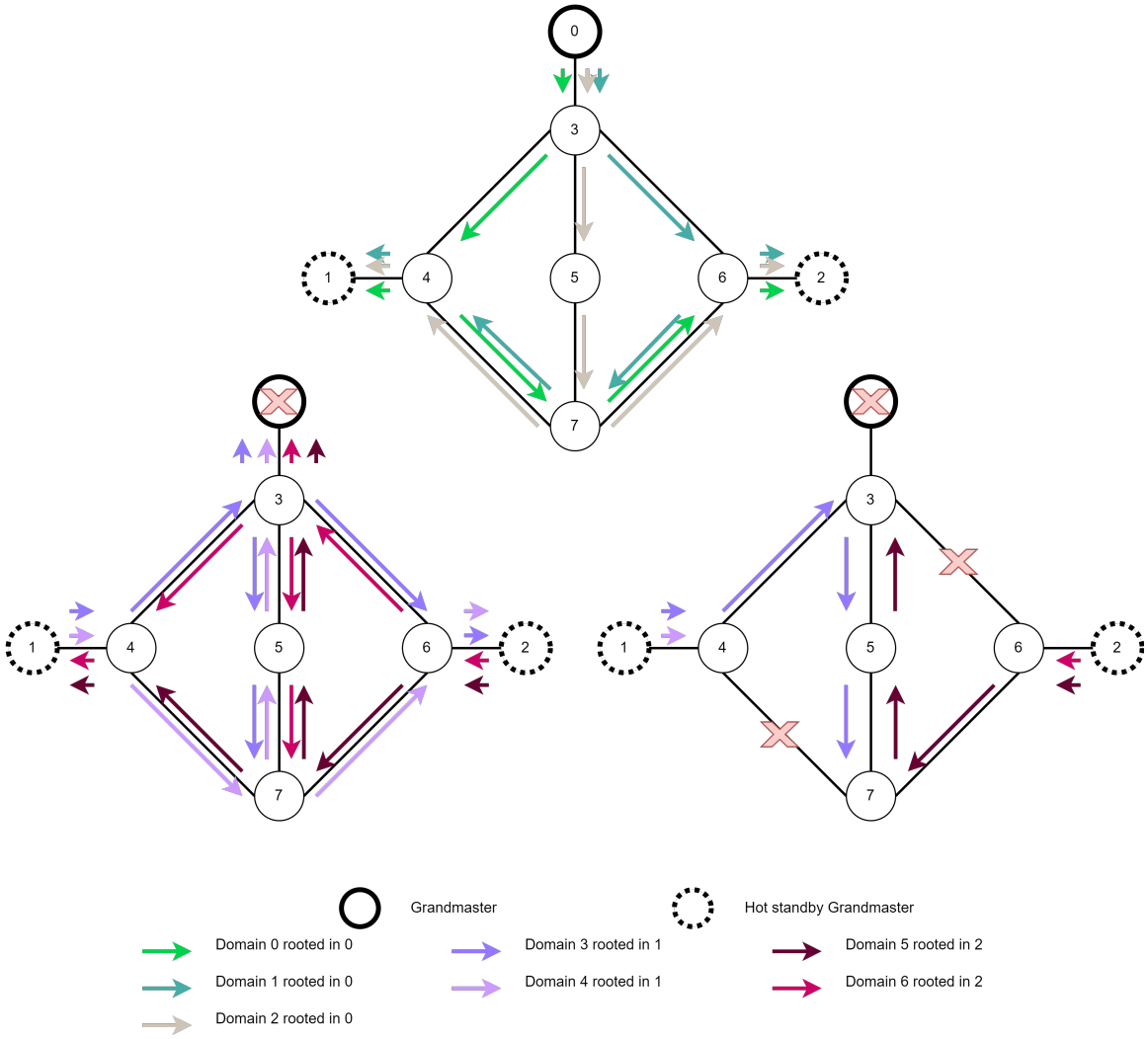


Figure 7.15: Illustration of the multiple time bases issue with 3 Grandmasters. On the top topology, only the part of the spanning tree that leads to the hot standby Grandmasters are display

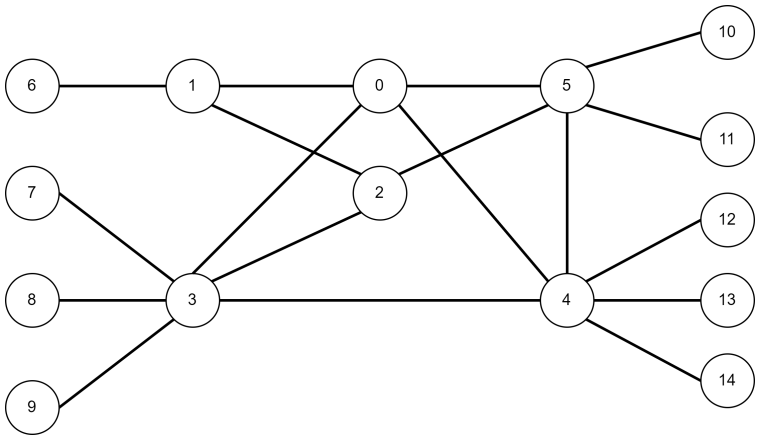


Figure 7.16: Digital audio cockpit network graph

are undergone, as shown in Figure 7.15-bottom right, we observe a case where some time-aware systems (such as node 1 and 4) uses a different time base than the other time-aware systems (such as node 2 and 6) in the network.

As in the case where we only have two Grandmasters, we can ensure that the remaining Grandmasters in hot standby have domains that cover all the existing routes between the different synchronization sources. In our example, it would be necessary that the hot standby Grandmaster 1 have 4 domains which cover all the routes to the hot standby Grandmaster 2 in order to synchronize the latter and avoid the appearance of a second time base. However, for this to work, the hot standby Grandmaster must be able to dynamically select its synchronization reference source. It could for example use a priority mechanism similar to the one we studied during our study of static reconfiguration mechanisms in the previous chapter. A priority system, for example based on the domain id, could be used to ensure the determinism of the solution. However, these are only hypotheses and the state of the draft of P802.1ASdm, at the time of writing these lines, does not decide on the support or not of a GM reference selection mechanism by the hot standby Grandmasters. We will therefore limit the use case of several Hot Standby Grandmasters to these few lines.

7.4.4 Results for the EDEN topologies

The number of domains required when applying this solution to the topology studied in the case of the EDEN project with a main Grandmaster, a hot standby Grandmaster and two domain per GM is presented in Table 7.14. The topology of the digital audio cockpit is recalled in the Figure 7.16. We observe that in the case of the automotive topology, only one additional domain is needed to avoid this problem. However, in the case of the AFDX network, which offers many more paths between two points, the number of additional domains required varies between 7 and 15 depending on the placement of the Grandmasters. Although the number of domains is higher with the second placement possibility of the grandmasters, these 15 additional domains that are spread over the two links to reach node 6 consume less than 0.01% of the bandwidth of these two links at 1Gb/s. However, such a multiplication of the number of messages could increase the latency of lower priority messages. This point will be analysed in the chapter 9.

Other solutions are feasible to ensure that the second condition is not possible but due to the large number of time-aware systems compared to the number of Grandmasters in hot standby, such

Topologie	Automotive	Digital audio	AFDX	
GM node id	0	9	3	1
Hot standby GM node id	6	14	4	6
Number of additional domains to cover all paths	+1	+6	+7	+15
Total number of domains	5	10	11	19

Table 7.14: Table presenting the number of domains necessary to avoid the appearance of a second time base according to the topology and the location of the Grandmasters. Two Grandmasters configurations are shown for the AFDX topology

a solution has a much higher implementation cost.

As in the previous section, preventing the occurrence of multiple time bases can make the system more robust than specified. Indeed, let's take the example of the automotive case study, this system must resist to a failure. On this topology, a single failure does not create the necessary conditions for the appearance of the second time base. Thus, it is possible not to try to solve this problem according to the requirements of the system. However, it is necessary to pay attention to the fact that this failure is fail-silent and may therefore require a non-standard monitoring mechanism to detect it, if it is not prevented by one of the solutions proposed above.

7.5 Execution time enhancements

The different points discussed in the previous sections are often based on an exhaustive exploration of all possible configurations in order to compare them. Exhaustive exploration can be quite time-consuming. In this section, propose optimizations to reduce the time complexity of these explorations.

7.5.1 Identification

The first step of this study is to identify time-consuming actions in order to optimize them. For that, we present the execution time of the points discussed previously for the automotive topology and the AFDX topology with 2 and 3 domains in the Table 7.15. All these measurements were done with a single-thread Python implementation and executed on an Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz.

With this table, we observe that the most time-consuming element is the inter-Grandmaster synergy evaluation lasting between 1 hour 20 minutes and 8 hours 25 minutes, followed by the robustness evaluation lasting between 14 secondes and 3 hours 10 minutes. In front of these execution times, the evaluation of the spanning tree precision or the analysis of the multiple time base issue consumes a negligible time lasting less than a second. The two main time-consuming mechanisms use the robustness metric in their evaluation. So to reduce the execution time of the method, we need to optimise the calculation of this metric.

7.5.2 Proposed optimisations

We assume that this metric takes a long time to calculate because of the evaluation space. Indeed, this space is the cartesian product of three sets : the set of nodes, over which we have to check the existence of a path to one of the Grandmasters, the set of possible failures, over which the metric

	Precision evaluation	Robustness evaluation	Inter-Grandmaster synergies evaluation	Multiple time base isssue checker
Automotive topology with 2 domains rooted in 0	0.04	14	4858	0.0002
Automotive topology with 3 domains rooted in 0	0.04	1822	15433	0.0002
AFDX topology with 2 domains rooted in 0	0.01	16	30213	0.00005
AFDX topology with 3 domains rooted in 0	0.01	11234	9463	0.00005

Table 7.15: Execution time (in seconds) of the different evaluations according to the topology and the number of domains. The evaluation of the synergy between Grandmaster takes place with two other domains whose root is node 4 for both topologies

iterates, and the set of spanning tree to evaluate. In what follows, we verify this hypothesis and propose a way of reducing each set in order to make the execution of the method less time-consuming.

Let's start by verifying the hypothesis using the execution times measured during the robustness evaluation stage. We observe a relatively similar execution time between the two 2-domain configuration while the number of spanning tree sets evaluated is very different: 66 sets for the automotive topology against 12561 for the AFDX topology. The number of failures studied is also very different, but this time it is the automotive topology that has the most failures, with 63 against 18 failures studied for the AFDX topology. Thus the AFDX topology, with its almost 200 times more spanning trees, is almost as quick to assess in terms of robustness as the automotive topology because it is assessed on a number of failures 3.5 times lower than the latter, thereby confirming our hypothesis.

Continuing this analysis, we can also observe that with three-domain configurations, compensation by the number of failures is no longer sufficient for the AFDX topology because of the explosion in the number of 3-spanning tree sets, leading the AFDX topology to an evaluation time 6 times slower than the automotive topology. Indeed, 2016 combinations of two failures are studied over a 220 set of 3-spanning trees in the case of the automotive topology against 171 combination of two failures on 657359 set of 3-spanning trees for the AFDX topology. This is also the case for the inter-Grandmaster synergy evaluation. Indeed, in the case of the 2-domain automotive configuration, 24 set combinations are evaluated using 41727 failure combinations while for the two-domain AFDX configuration, 256036 set combinations are studied 987 failure combinations.

We therefore propose two methods for reducing the search space. The first is to reduce the number of nodes to iterate on and the second is to reduce the number of spanning trees.

Let's start with the first proposal, which is based on reducing the number of nodes. This optimisation has a positive side-effect. By reducing the number of nodes, we also reduce the number of links used to connect them to the network. This double reduction reduces the number of possible failures. Indeed, there can no longer be a failure on a link or node that no longer exists. However, not every node can be removed. Nevertheless, when comparing the two topologies, there is one major difference. This difference between the AFDX topology, where the number of failures is low, and the automotive topology where the number of failures is higher, is the presence of end-nodes.

Indeed, in the case of the automotive topology, they represents 46 failures out of 63 during the two domains robustness evaluation. However, on this topology, whether a link between the switch and the end node or the end node suffers a failure, the result is the same regardless of the spanning tree used. It is therefore possible to ignore these failures by removing this end node in order to reduce the number of failures and node to evaluate. The execution times after the implementation of this mechanism are displayed in the table 7.16. In the automotive case, a reduction of the execution time between 71.4 and 99.2% is measured on the two mechanisms using the robustness metric.

	Precision evaluation	Robustness evaluation	Inter-Grandmaster synergies evaluation	Multiple time base isssue checker
Automotive topology with 2 domains rooted in 0	0.04	4	147	0.0002
Automotive topology with 3 domains rooted in 0	0.04	177	127	0.0002

Table 7.16: Execution time (in seconds) of the different automotive evaluations according the number of domains after reducing the number of failure investigated. The evaluation of the synergy between Grandmaster takes place with two other domains whose root is node 4 for both topologies

This impressive reduction of the execution time observed on the automotive topology is null for the AFDX topology. Indeed, the latter has no end node in our study, which focuses on the core network. To reduce the execution time observed with this topology, we need a second optimisation. This optimisation aims to reduce the number of spanning trees to be evaluate. Therefore, we propose the following heuristic: multiple spanning trees cannot coexist on the outgoing link of the Grandmaster if the topology and the number of domains allow it. Indeed, if an end-node is directly connected to Grandmaster 0 of the AFDX topology, then several domains will have to cohabit on this link. Another limitation, if we study a 4 domain configuration, the Grandmaster being connected with only three links to the rest of the network, the fourth domain will have to share one or more links with the first three domains. An implementation using this heuristic reduce the robustness evaluation on the AFDX 2-domains configuration from 16s to 8s by reducing the number of set from 12561 to 4321 and the 3-domain one from 3h07 to less than 9 minutes by reducing the number of set from 657359 to 24389 without changing the output. However, this heuristic is not applicable when evaluating the synergies between Grandmasters because the selected sets are already the most robust for their respective Grandmasters.

7.6 A complete methodology

The above results can be put together in a method that is intended to be followed by a network designer to determine which static configuration to use. The method proposed and illustrated in Figure 7.17 is detailed in this section. This section concludes with an application to two case studies from the Eden project.

7.6.1 Design methodology

The objective of this method is to determine the optimal configuration of spanning tree i.e the most precise configuration among the most robust ones. This method is composed of 5 steps as

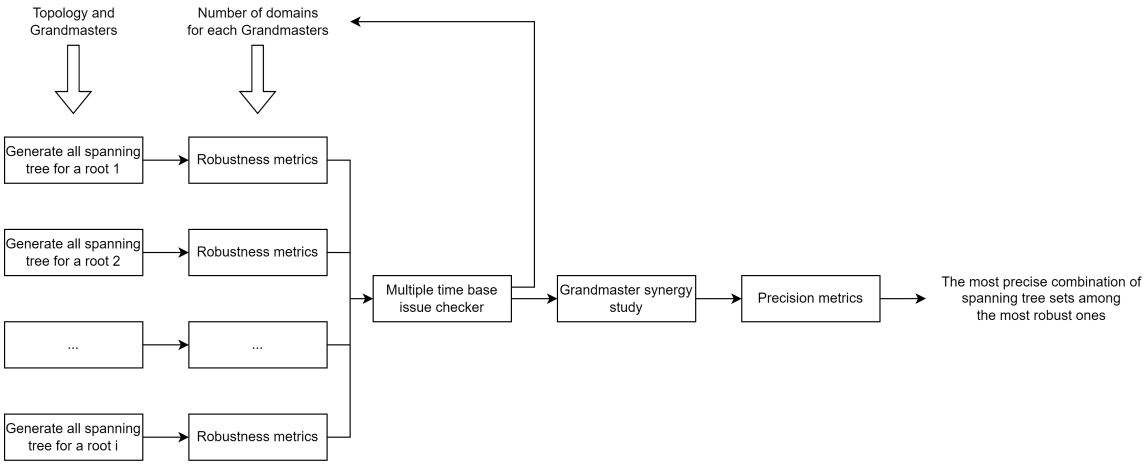


Figure 7.17: Illustration of the proposed method to determine the best static configuration.

detailed below. The only inputs required are the topology, the Grandmasters used and the number of domains required for each Grandmaster. The following describes the method step by step.

The first step of the method, called "Generate all spanning tree for root i" in Figure 7.17, simply consists in finding the all possible spanning trees for the Grandmaster i. The second step aims at selecting the most robust spanning tree sets for the Grandmaster i. To do this, the sets of k spanning tree are studied with the help of the robustness metric presented in 7.1.2. Here, k is the number of domains used for the studied Grandmaster. Only the sets with the best results are selected for the next steps. These two first steps are repeated for the different Grandmasters of the network.

Based on the work presented in 7.4, the third step checks if the multiple time base problem is possible with this configuration. If it is not possible, the rest of the method can be executed directly. If it is possible then two options are available to the designer, the first one is to go back to the previous step by changing the domain number input in order to cover all the possible paths with spanning tree between the different Grandmasters. The second option is to go to the next step of the method by adding domains that are not spanning trees, but only cover the missing paths between the Grandmasters. In this last case, the added domains are ignored in the following evaluation because they do not cover the whole network.

The next step is the study of the synergies between the domains of the different Grandmasters, as explained in section 7.3, in order to reduce the options for the next step. Thus, each combination of sets is studied using the robustness metric. As for the second step, only the combinations having obtained the best score are selected for the next step.

Finally, the last step allows us to sort out the different combinations of set according to their precision evaluation using the metric selected in 7.1.1. To be more specific, the precision score is calculated for each spanning tree of the combination of set. The worst score is assigned to the combination of set to rank them. As before, only the best are kept. Then we repeat the ranking with the second worst score of the remaining overall combinations of set in order to eliminate the worst. We repeat this until only one set combination remains or after ranking by all the precision scores of these set combinations. The combination(s) then selected is(are) the most precise(s) among the most robust overall combination sets.

7.6.2 Results for the automotive and AFDX use cases

Let's apply this method on the topology of the automotive case study. In this example, the network designer wants a configuration that can withstand one failure (where possible) and he decides to use a configuration with a Grandmaster on node 0 and a hot standby Grandmaster on node 4 with two domains each.

He applies the first step and generates the 12 possible spanning trees for each of Grandmaster. Then he uses the robustness metric to study the 66 combinations of 2 spanning trees for both Grandmasters. This first ranking allows him to select the 6 best spanning tree sets (as detailed previously in the Table 7.10) for Grandmaster 0 and the 4 best ones for Grandmaster 4. He then looks at the multiple time base issue. With this configuration, the method tells him that it would need three domains for the Grandmaster 0 to always stay in contact with the hot standby Grandmaster 4. The designer then chooses to stay with a two full domain configuration for GM 0 and only adds a third domain which will cover the missing path. Then comes the study of the synergy between the four domains. This second classification allows the designer to reduce the 24 possible combinations between the 6 and 4 most robust spanning trees to the 8 combinations ranked first in the previously presented table 7.13. These most robust combinations are then ranked according to their precision score. After this step, only one combination, spanning trees 2 and 9 from Grandmaster 0 combined with spanning trees 0 and 3 from Grandmaster 4, remains as the most precise among the most robust ones.

In the case of the more connected AFDX topology, with node 3 and node 4 as Grandmaster with 2 domains each, the method helps the designer to narrow down from 157 778 721 to 1 combination (spanning trees 7 and 47 rooted in node 3 and spanning trees 2 and 106 rooted in node 4) which are the most precise among the most robust spanning tree set combinations.

7.7 Configuration optimisation

In addition to providing a method for determining a set of spanning trees that meets the needs of the system, this chapter also offers keys to optimize networks for synchronization thanks to the metrics and other results. In this section, we propose and illustrate some possible optimizations that we have applied to the case studies of the EDEN project.

In the rest of the chapter we started from a topology and looked for the optimal spanning tree sets, but the metrics can also be used earlier in the designe phase. Some possible uses are listed below.

- Evaluation of different topologies in order to select the best in terms of precision and/or robustness.
- Grandmasters placement optimisation
- Enhancement of an existing topology by adding (inexpensive) links.

These last two examples are illustrated in the remainder of this section.

Let's start by improving a topology using inexpensive additions. To demonstrate this capability, we set out to improve the precision and the robustness of the automotive topology by adding a single link. To do this, we iterate over all the 464 possibilities of a new link by calculating the score obtained by the metrics. For this example, we assume that the Grandmaster is node 5. The results obtained are presented in the Table 7.17.

New link	Best precision score	Best robustness score
(0, 1)	74	157
(0, 2)	74	153
(0, 3)	74	157
(0, 4)	74	157
(0, 5)	74	131
(0, 6)	72	157
(1, 2)	74	153
(1, 3)	74	157
(1, 4)	74	157
(1, 5)	67	114
(1, 6)	74	157
(2, 3)	74	153
(2, 4)	74	153
(2, 5)	70	123
(2, 6)	74	153
(3, 4)	74	157
(3, 5)	64	114
(3, 6)	74	157
(4, 5)	70	114
(4, 6)	74	157
(5, 6)	70	114

Table 7.17: Best precision and robustness score according to one new link addition to the automotive topology. Only the results for the new links between the switches are presented since adding a link between a switch and an end system provides similar results for each combination.

Using these scores, we observe a gain in terms of precision and robustness by adding a link between node 5 and nodes 1 (scores : 67 and 114) or 3 (scores : 64 and 114) or 4 (scores : 70 and 114) or 6 (scores : 70 and 114). In terms of precision, adding one of these links makes it possible to bring many nodes closer to the Grandmaster, especially with the link (3,5). For robustness, adding a link from the Grandmaster to one of these nodes makes it possible to have two exit paths from the Grandmaster but above all allows access to switches offering several independent paths to the rest of the network. For new links where one of the nodes is an end station, the results are very similar for all combinations. Indeed, the end stations can only receive synchronization information, it cannot serve as a relay for this information like the switches. Thus, in terms of precision, the gain does not propagate to devices close to this end station. In terms of robustness, the end station has two paths to contact it, which makes its access more robust, but as for precision, this improvement has no effect on the neighbourhood. Thus, in this case we would recommend adding a link between nodes 5 and 1.

Metrics can also be used to optimize the placement of Grandmasters to maximize precision and/or robustness. Let's take again the example of automotive topology. By iterating over all the nodes and calculating the score for the two metrics, we obtain the rankings presented in the table 7.18. With these results, we observe that placing the Grandmaster on node 0 is the best option (scores : 53 and 115), which allows both to choose between the most robust and the most precise spanning trees. Switches 1,2,3,4 and 5 obtain a less good result (scores : [64;76] and [121;161]) for the precision metric because passing through node 0 or node 6 is mandatory to reach the other

nodes. Switch 6 (scores: 77 and 127) is the worst switch option in terms of precision because of the long paths needed to reach switches 2 and 5 and their end-nodes. In terms of robustness, after the switch 0, there are two groups of switches. The first group in the ranking, composed of nodes 1,3,4 and 6, obtains the best robustness score thanks to their two links to the core network. Unlike the second group, made up of switches 2 and 5, which suffers from their single point of failure towards the core network. It should be noted that we observe some difference in score in switches of the same group because of the number of end-nodes connected to them. As for the scores when an end-station is used as a Grandmaster, we observe that the switches remain better in terms of precision because one less hop is necessary and in terms of robustness because of one less single point of failure. Of course, there is a difference in score between the end-nodes depending on the switch to which they are connected. Thus, if an end-station must be used to be GM then the best option in terms of precision and robustness remains to connect it to node 0.

The Grandmasters placement can also be optimized for another criteria : the number of domains. In this last exemple, we considere an optimization of the placement of classic and hot standby Grandmasters in order to minimize the number of domains necessary in order to prevent the appearance of concurrent time bases. Indeed, we saw in the section 7.4 that the number of domain necessary for the AFDX topology varies between 9 and 17. By calculating the number of domain necessary for each combination of GM / HSTby GM, it is therefore possible to determine the combination requiring the fewest domains. Note that the choice of GM placement can also be limited by other aspects, such as access to a GPS antenna. Without taking into account these other constraints, such an optimization on the AFDX topology find 12 combination that need 9 domains, such as GM on node 6 and HSTby GM on node 4 or GM on node 5 and HSTby GM on node 6.

Conclusion

In this chapter, we solve the difficulties of the design phase of the static configuration raised in the previous chapter by proposing a method allowing to determine the most precise combination of sets among the most robust ones. This optimum meets the needs of the EDEN project, but may not be suitable for all needs. To take this case into account, the method remains highly configurable by playing on the call order of the metrics. However, this method relies on brute force by evaluating many combinations or sets using metrics. It is therefore interesting for small embedded networks but becomes very time-consuming in the case of a more connected and extended network due to the explosion in the number of spanning trees possible on such a topology. We also offer several solutions to the multiple time base issue, including one based on the use of additional domains which is integrated into the method. In addition to this method, we propose to explore the possible use of the results of this chapter for purposes of topology and/or placement of Grandmasters optimization. However, these few proposals open up numerous perspectives for topology optimisation in order to make synchronisation precise and/or robust.

Root	Best precision score	Best robustness score
0	53	115
1	70	124
2	76	161
3	64	121
4	76	127
5	74	157
6	77	127
7	82	173
8	82	173
9	82	173
10	82	173
11	99	182
12	99	182
13	99	182
14	99	182
15	105	219
16	105	219
17	105	219
18	120	233
19	91	175
20	93	179
21	93	179
22	93	179
23	93	179
24	93	179
25	105	185
26	103	215
27	103	215
28	103	215
29	103	215
30	106	185

Table 7.18: Table presenting the best score obtained with the precision and robustness metrics with two domains for each of the possible Grandmasters.

Part IV

Synchronization and the other network activities

Chapter 8

Impact of synchronization on bandwidth

The Time-Aware Shaper (TAS) [27] is a mechanism that completely relies on synchronization for optimal operation. Moreover, its deployment requires a more constrained synchronization precision than the one needed by distributed applications. It is therefore an important element of the precision dimensioning. In this chapter, we are interested in the requirements of this mechanism in order to dimension at best the parameters impacting the precision and the gPTP traffic. We show that the loss of bandwidth related to the over-dimensioning of TAS windows (with time guard bands) to adjust to the synchronization precision is small to negligible. This allows us to conclude that the need for sub-microsecond precision is often over-dimensioned for TAS.

8.1 Time guard band

The Time-Aware Shaper relies on the scheduled opening and closing of the FIFO queue in the egress port of a network device. It can be used to minimize the latency or jitter of very time constrained flows. The period of opening of one or several FIFOs is called a TAS window. A sequence of several windows represents a schedule called Gate Control List (GCL) in the standard. However, as the precision is not perfect, it is necessary to take into account the desynchronization of the different time-aware systems in the dimensioning of the windows. Otherwise, messages could arrive before or after the opening of the window dedicated to them as illustrated in green in Figure 8.1. To take into account this potential desynchronization, a time guard band is added at the beginning and at the end of the window. Its duration is equal to the maximum desynchronization between the two time-aware system as illustrated in purple in Figure 8.2. This maximum desynchronization is given by the an upper bound on the worst-case precision.

The addition of these time guard bands is then equivalent to over-dimensioning the TAS windows and implies a loss of bandwidth. This loss of useful bandwidth is proportional to the precision. If the synchronization is not precise then the guard bands are large and the lost bandwidth is important. One option to make the synchronization more precise is to synchronize more often but this consumes more bandwidth. In the following, we study this problem using the satellite use case of the EDEN project. We chose this specific use case because it requires the deployment of TAS to keep time-triggered operations similar to the ones provided by the MIL-STD-1553 bus in today's satellites.

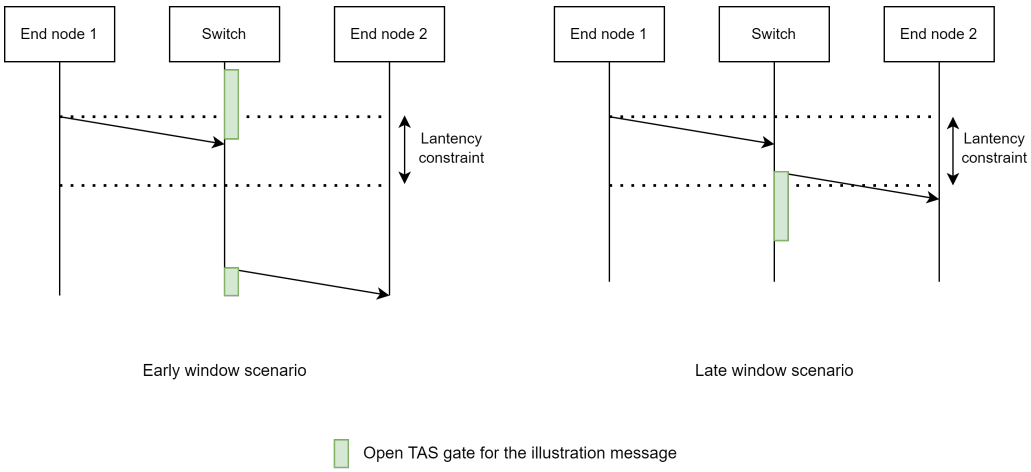


Figure 8.1: Impact of desynchronization on TAS: on the left panel, the message arrives after the gate closes in the switch while on the right panel, the message arrives before the gate opens, and thus has to wait for being served.

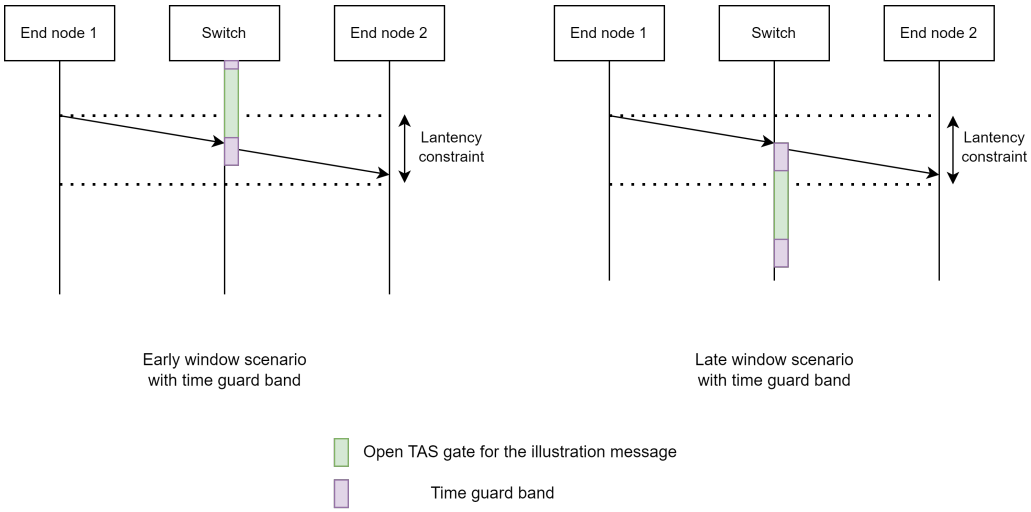


Figure 8.2: Benefit of the time guard band on TAS transmissions that compensates for the largest desynchronization of the time-aware systems.

8.2 Bandwidth consumption

As previously stated, in the satellite use case, the architecture is centralized around the On Board Computer (OBC). The OBC issues commands that must be executed by the target systems as soon as the message is received. These messages must thus be received with a very low jitter to ensure deterministic command execution. The applications are not synchronized but the sending and receiving of the command with a very low jitter allows several simple capacity-limited and decentralized systems to act at the same time. This choice is based on legacy equipment and could allow a smooth transition from MIL-STD-1553 to TSN without bringing major changes to existing on-board applications. Thus, this mode of operation induces an important number of TAS windows in order to minimize jitter. In the following, we focus on the link connecting OBC A to SWITCH PL A because all the flows requiring low jitter are transmitted on this link. In the direction OBC A $\rightarrow$ SWITCH PL A, one of the schedules envisioned in the project and allowing to satisfy the jitter constraints is described in Table 8.1. We observe a cycle of 125ms containing 23 windows (i.e. 184 windows per second) allowing to alternate between the FIFO of the *CCUltraLowJitter* messages and the other FIFOs.

Start time (ms)	End time (ms)	CCUltraLowJitter FIFO	All other FIFOs
0	50	closed	open
50	50.002	open	closed
50.002	95	closed	open
95	95.002	open	closed
95.002	95.25	closed	open
95.25	95.252	open	closed
95.252	95.5	closed	open
95.5	95.502	open	closed
95.502	95.75	closed	open
95.75	95.752	open	closed
95.752	96	closed	open
96	96.002	open	closed
96.002	96.25	closed	open
96.25	96.252	open	closed
96.252	96.5	closed	open
96.5	96.502	open	closed
96.502	96.75	closed	open
96.75	96.752	open	closed
96.752	97	closed	open
97	97.002	open	closed
97.002	97.25	closed	open
97.25	97.252	open	closed
97.252	125	closed	open

Table 8.1: TAS scheduling for the OBC A $\rightarrow$ SWITCH PL A link allowing to respect the latency constraints of all the flows crossing this link

Figure 8.3 illustrates the percentage of bandwidth consumed by the time guard windows as a function of synchronization precision with the 184 windows per second of the satellite use case. On this figure, we can see that a precision of 3ms would lead to a dimensioning of the time guard bands

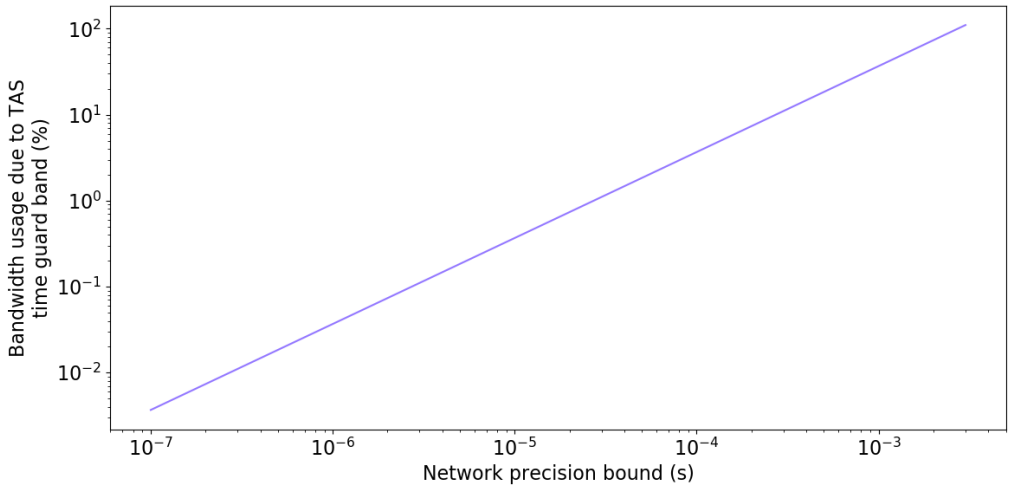


Figure 8.3: Bandwidth used by the time guard band as a function of the synchronization precision with 184 TAS windows per second.

that would consume more than 100% of the bandwidth. At 300µs, this still represents a substantial proportion, with a consumption of 11%. At 30µs, this represents a little over 1%. This figure illustrates the importance of tightly bounding worst-case precision. In what follows, we'll focus on the left side of the graph, taking advantage of our model for computing a tight precision bound on this use case, while also taking into account the bandwidth consumed by synchronization protocol messages.

Let's assume that all the devices of the network behave in the same way in terms of synchronization and that the parameters impacting the precision are those described in Table 8.2. It leads to a precision between -1.5µs and 1.46µs based on our 1000Base-T bound derivation. However, this precision bound is calculated between the Grandmaster and a time-aware system. Thus a time-aware system may experience a positive drift close to the upper bound and another one may experience a negative drift close to the lower bound. The precision between any two time-aware systems is then bounded by the following equation:

$$P_{network} = |min(P_i^L)| + |max(P_i^U)| \tag{8.1}$$

In our case, with the assumption that the time-aware systems are identical, we obtain a bound of 2.96µs on the precision between any pair of time-aware systems in our network. By adding this precision at the beginning and the end to the 184 windows of the studied link, 1.089ms are not used per second or 136160 bytes per second in the case of a gigabit link.

Although small, this loss can be reduced by increasing the precision. To do so, the simplest way is to double the frequency of the `Sync` but at the cost of an increase in the bandwidth consumed by the synchronization protocol. This trade-off is studied in Figure 8.4 which illustrates the bandwidth loss caused by the time guard bands added to the bandwidth consumption of the gPTP messages for different amount of TAS windows per second. This bandwidth consumption is calculated using Equation (6.2) presented in chapter 6 for a static 1-domain configuration with the message frequencies of Table 8.2. The considered data rate is 1Gb/s.

On this figure, we can see that the bandwidth consumed increases with the number of windows

Parameters	Value
I_s	0.125s
I_p	1s
ρ_{GM}	0.02ppm
ρ_i	10ppm
τ	1ms
G	10ns
d^{min}	200ns
A	6.85ns
$J_{GM \to L}$	29.7ns
$J_{L \to GM}$	8ns
J_{fup}	2ms
i	3

Table 8.2: IEEE802.1AS parameters used to calculate the bounds on precision for the spatial use case.

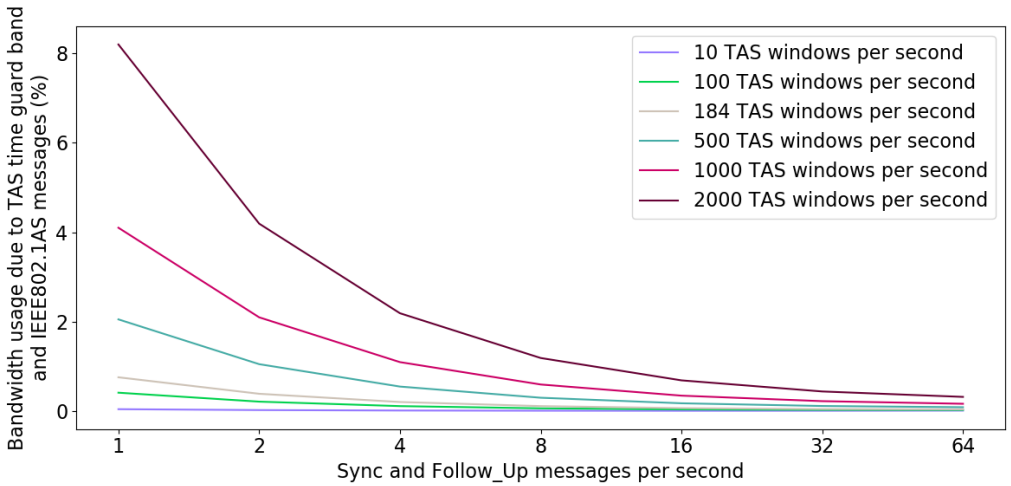


Figure 8.4: Bandwidth used by the time guard band and by the synchronization messages as a function of the synchronization frequency for different number of TAS windows per second.

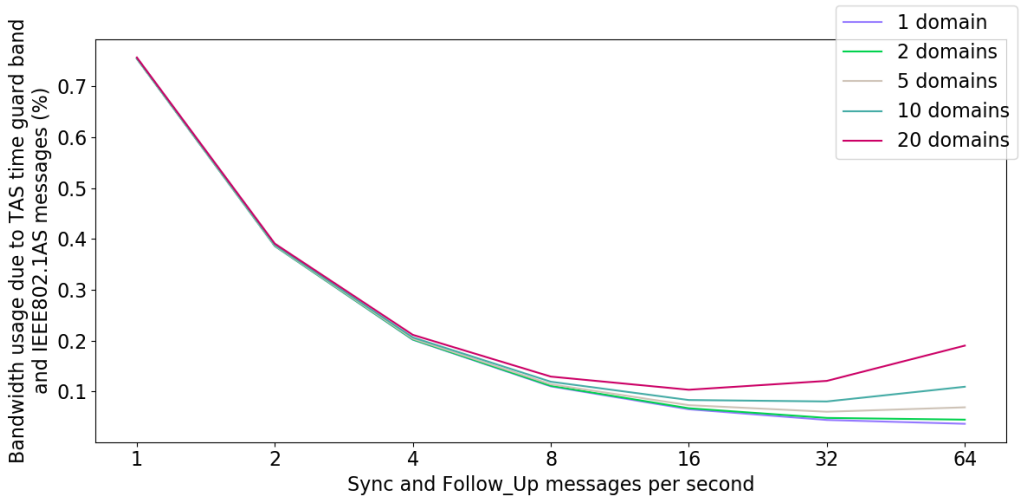


Figure 8.5: Bandwidth used by the time guard band of 184 TAS windows per second and by the synchronization messages as a function of the synchronization frequency and for different numbers of domains.

due to the larger number of time guard bands. The increase of the precision using the *syncInterval* allows to reduce the bandwidth lost by the time guard bands at the cost of a negligible additional bandwidth consumption in this example. However the gain is significant for the first *syncInterval* reductions, then quickly reaches a plateau. This plateau is caused by the fact that, as illustrated in chapter 5 with Figure 5.8, reducing the *syncInterval* does not allow to reach a perfect precision due to other source of error like the granularity or the physical jitter. This induces that the bounds stagnates, which implies a stagnation of the size of the guard bands causing the plateau observed on the figure.

For our use case of 184 TAS windows per second, even with one `Sync` per second, the bandwidth consumption is less than 1%. For a `Sync` interval of 125ms, the bandwidth consumption is of 0.11%. In this case, doubling the frequency of the `Sync` allows to reduce the consumption to 0.06%. Since the bandwidth consumption is always very low (less than 1% of a gigabit link), we do not think it is necessary to try to improve the synchronization precision if the dimensioning need is inherited from TAS, unless TAS is used to reduce flow jitter. However, in the more prospective case of future networks relying on more TAS windows per second, improving the precision to increase the usable bandwidth may make sense. Indeed, in the example at 2000 windows per second, synchronizing every second means losing 8.2% of the bandwidth. In this case, synchronizing more often brings a real gain. We compute a loss of 1.2% (respectively 0.7%) when synchronizing every 125ms (respectively 62.5ms).

Nevertheless, we have seen in part III that one domain is not enough to allow a robust synchronization. Indeed, we have observed that between two and about twenty domains are needed for our case studies. However an increase in the number of domains means an increase in bandwidth. Figure 8.5 illustrates this increase in bandwidth usage as a function of the number of domains crossing the link in the same direction with the same parameters as the previous figure but this time focusing only on the 184 TAS windows per second configuration. In this figure, we can see that despite the 20-fold increase in the number of domains, the overall bandwidth consumed remains negligible. But,

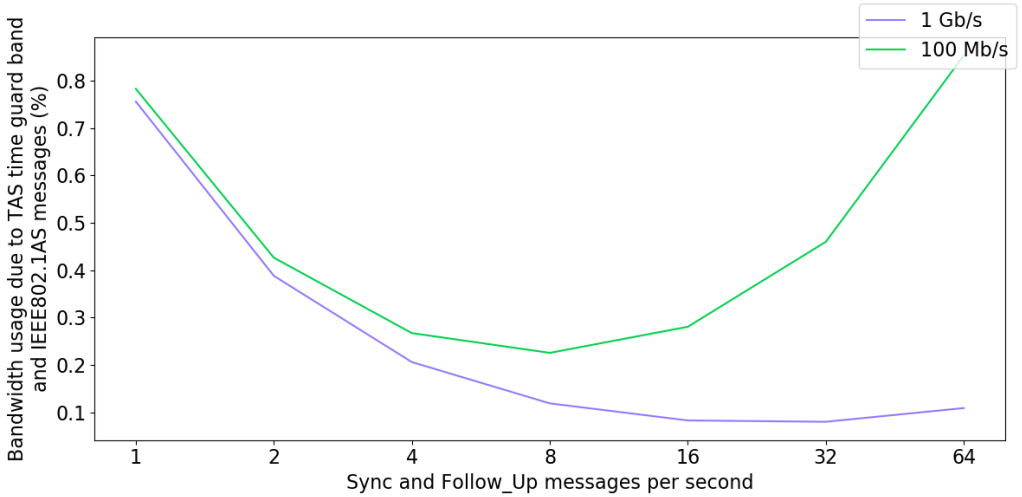


Figure 8.6: Bandwidth used by the time guard band of 184 TAS windows per second and by the synchronization messages as a function of the *syncInterval* for different physical layer with 10 domains.

this increase in the number of domains shows that starting from a *syncInterval* threshold the precision costs more in bandwidth usage because of the bandwidth consumed by AS messages than what it allows to gain by reducing the duration of the time guard bands. For example, we observe that the lowest bandwidth consumption occurs for a *syncInterval* of 0.03125 ms (respectively 0.0625ms) for the 10 (respectively 20) domain configuration. This optimum also exists for configurations with less domains but would appear for even smaller *syncInterval* values. Such *syncInterval* values are rarely supported by TSN devices and the gain obtained remains very low compared to a running *syncInterval* values like 0.125ms or 0.0625ms.

We also saw in part II that the precision depends on the physical layer used. Figure 8.6 illustrates the relation between the bandwidth used by the TAS time guard bands and the synchronization according to the *syncInterval* and the physical layer. In this figure, we observe a much higher bandwidth usage when 100Mb/s is used. There are two reasons for this. The first one is that the bounds are larger when using 100Base-T compared to 1000Base-T as illustrated in part II. The guard bands are therefore longer and waste a larger percentage of the bandwidth. The second cause is related to the use of bandwidth by synchronization messages. Let's take the example of a `Sync`. This message is 64 bytes long. If it is sent 8 times per second on a 1Gb/s link, it represents 0.0004% of the bandwidth used. On a 100Mb/s link, ten times slower, it represents 10 times more bandwidth used, i.e. 0.004%. So even though the messages exchanged and their periods are identical on both physical layers, the percentage of bandwidth consumed by the `Sync` messages is 10 times higher for 100Mb/s than for 1Gb/s resulting in a consumption that increases very quickly with the frequency of the `Sync` messages, as we observe when the *syncInterval* is smaller than 0.125s, but also with the number of domains.

8.3 Dimensioning by constraint

In this chapter, we have seen that the use of TAS has an impact on the usable bandwidth. This impact can be reduced by tuning IEEE802.1AS configuration parameters. In this section, we put into practice the observations of the previous section in order to configure the synchronization protocol while respecting a usable bandwidth constraint.

Let's illustrate this dimensioning on a fictional example with a high demand for TAS windows. Let's take the example of a 5-hop network with a lot of control/command messages. These messages being small, the physical layer interfaces used in this network are all at 100Mb/s. Moreover, these messages being very constrained in latency and jitter, our network needs 1000 TAS windows on the most constrained link. On this network, synchronization is constrained by the use of TAS. Indeed, distributed applications require millisecond precision synchronization. We also assume that, following the propositions of chapter 7, 4 domains are required to meet the resuired robustness guarantees. The constraint set by the network designer is that 99% of the bandwidth be usable.

The designer has the choice between 3 switches for his network. These switches are equipped with clocks of different qualities:

- Switch A : 50ppm
- Switch B : 25ppm
- Switch C : 10ppm

Based on these parameters, we can determine the percentage of non-usable bandwidth for the different switches. The results are displayed in Figure 8.7. On this figure, we observe that only switches B and C allow to respect the constraint of 99% of usable bandwidth in the studied frequency interval of the `Sync`. We also observe a minimum bandwidth consumption for both switches for a *syncInterval* of 0.03125s. This means that a smaller *syncInterval* than 0.03125s does not reduce the bandwidth usage but on the contrary increases it. Moreover, we can deduce from the calculations underlying this figure that in this example and in spite of the high demand in TAS windows, a precision under the microsecond is not always necessary to answer the need of such TSN network. Indeed, with the switch C and a *syncInterval* of 0.125s, the precision between the Grandmaster and any other time-aware system is upper bounded at 1.9µs for a lost bandwidth of only 0.8%.

Conclusion

In this chapter, we have determined the trade-off between the bandwidth wasted by the TAS time guard bands and the synchronization precision. We have studied the influence of the precision, the number of windows, the number of domains and the physical layer on the useless bandwidth. In other words, we observed an overall increase in bandwidth consumption with the number of TAS windows and domains, a higher consumption with 100Base-T compared to 1000Base-T and the presence of a minimum consumption depending on all the parameters mentioned above. However, all these impacts on the amount of lost bandwidth are relatively small. For example, with the default gPTP configuration and despite 2000 TAS windows per second, the bandwidth loss is less than 2%. Furthermore, we've shown the importance of tightly bounding precision to reduce wasted bandwidth, but also that synchronization with sub-microsecond precision is sometimes unnecessary, even when using highly constraining mechanisms such as TAS with a large number of windows per second.

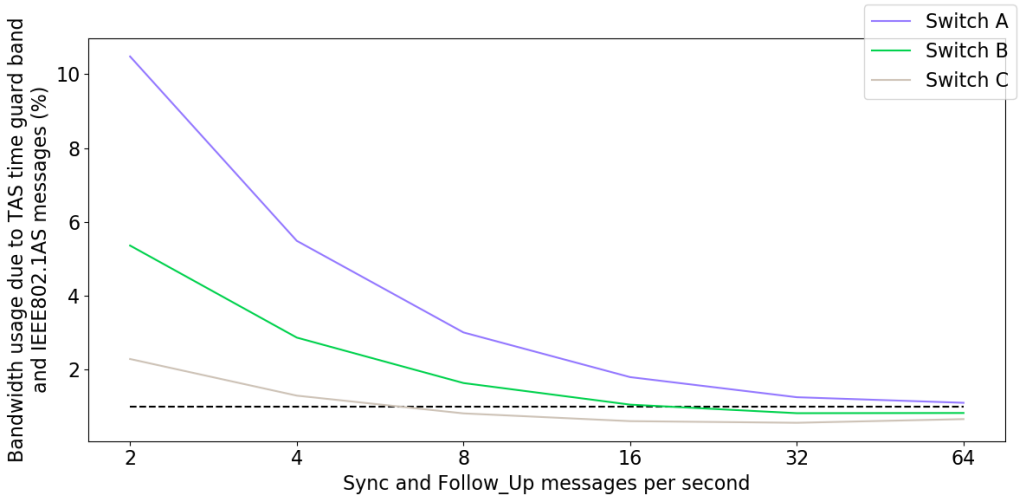


Figure 8.7: Bandwidth used by the time guard band of 1000 TAS windows per second and by the synchronization messages as a function of the synchronization frequency for different clock quality. Dashed black line represent the 1% constraint

Chapter 9

Impact of synchronization on other traffic

In this chapter, we study the impact of synchronization messages on the latency of applicative flows crossing the network on the satellite use case. Indeed, such flows must guarantee compliance with latency constraints for the proper operation of the applications for which they are intended. In this chapter we study the impact of synchronization on the worst-case latency bounds of applicative flows. We highlight the very small impact by testing as a function of the variation in synchronization protocol configuration increasing the precision and robustness of synchronization.

This study is based on the observation that synchronization messages consume very little bandwidth, as highlighted in the previous chapter. However, we also saw in chapter 5 that synchronization precision is highly configurable by playing with the *syncInterval* and in chapter 7 that the level of robustness to failure is also a function of the number of domains. Thus, a precise and robust configuration could lead to a high `Sync` and `Follow_Up` frequency that will be duplicated on numerous domains, thus counteracting the low message consumption. We will therefore study the impact of these small but numerous messages on the worst-case latency bounds of applicative flows.

9.1 Experimental setup

This study is carried out using a tool called Timaeus-Net, illustrated in Figure 9.1. This is a tool for network design and formal analysis of latency, jitter and memory usage in the network based on Network Calculus, developed as part of the EDEN project. It was developed to explore and analyse the impact of different TSN mechanisms on the worst-case bounds of flows of EDEN use cases.

Once again, we leverage the satellite use case since it is the one with the most constrained network traversal latencies and jitter. The study is carried out on all 102 flows, but only a representative subset is presented here. These flows belong to 13 traffic classes. These 13 traffic classes have been mapped onto the 8 Ethernet priority levels. TSN shaping mechanisms have been assigned to these 8 priority levels. Although configured, priority level 0 is not used for this case study. The shapers assignement to priority levels is the following one:

- Priority 7 : TAS + Static Priority
- Priority 6 : TAS + Static Priority
- Priority 5 : TAS + Static Priority
- Priority 4 : TAS + Static Priority

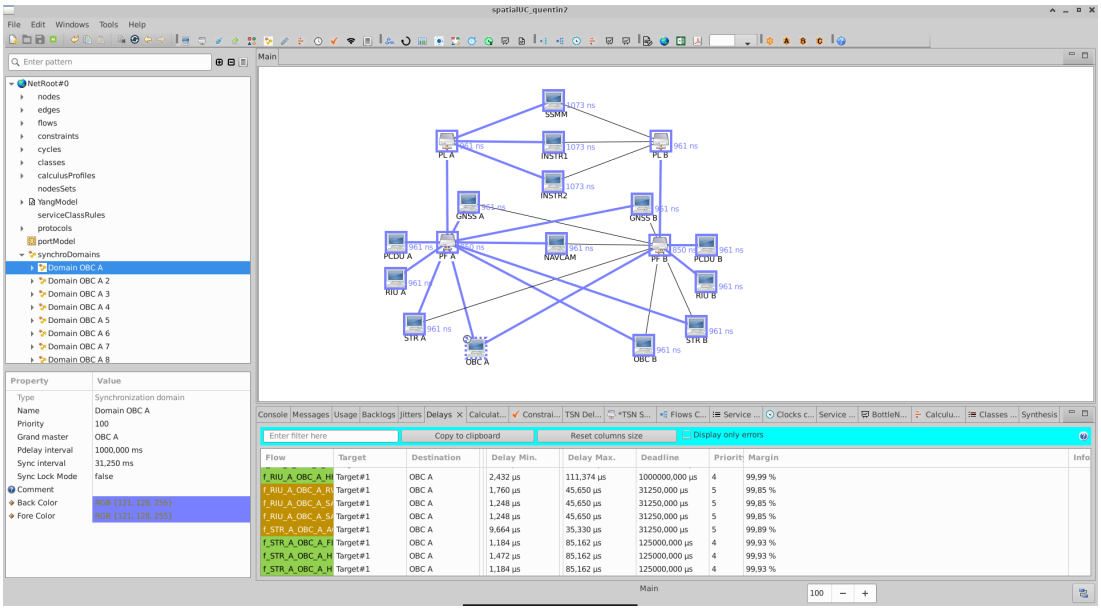


Figure 9.1: Screenshot of Timaeus-Net

- Priority 3 : TAS + CBS + Static Priority
- Priority 2 : TAS + Static Priority
- Priority 1 : TAS + Static Priority
- Priority 0 : TAS + Static Priority

Note that the 8 FIFOs of the different output ports have a Time-Aware Shaper assigned to them. The policy chosen for the TAS schedule is called exclusive gating. When the TAS gives access to the link to the priority 7 FIFO, all the other FIFOs are closed, and when the priority 7 FIFO is closed, all the others are open. This shaper assignment is explained by the need to precisely control the jitter of priority 7 flows and the large volume of priority 3 frames (there frames carry the output of earth observation instruments), which are only lightly constrained in terms of time. Using a CBS on the latter enables the network to control the rate at which these frames are sent, thus reducing latency on lower-priority flows. For other flows, access to the medium is arbitrated using the static priority mechanism derived from classical Ethernet. So if the priority 5 FIFO needs to send a frame, it will have to wait until the priority 6 FIFO is empty.

Timaeus determines a configuration for these different shapers (scheduled windows and CBS slope value) which satisfies the needs of each flow in terms of worst-case traversal latency and jitter.

In addition, it should be noted that all flows are duplicated on two independent and symmetric paths using the Frame Replication and Elimination for Reliability(FRER) mechanism. To be more specific the replication is done at the transmitting end-station and the elimination at the receiving end-station. So for our study, we focus on just one side of the network.

In terms of synchronization, following the conclusions of Chapter 5 that quantified the impact in terms of worst-case precision and network jitter suffered by `Sync` and `Follow_Up`, we place gPTP messages at priority level 4. It gives priority 6 and 5 messages greater transmission opportunities, reducing their latency in the worst-case scenario.

Using this design and analysis environment, the influence of three IEEE802.1AS parameters on

worst-case flow latencies is studied as follows: in Section 9.2, we study the impact of the sending period of `Sync` messages, i.e. *syncInterval*; in Section 9.3, the study focuses on the impact of the number of domains and of the Common Mean Link Delay Service (CMLDS). The latter mechanism enables the propagation delay measurement mechanism to be executed in a single domain and the result passed on to the other domains.

The spanning tree used to distribute synchronization information is the one illustrated in the Timaeus-Net GUI in Figure 9.1. The Grandmaster is OBC A. The port state configuration mechanism is static port configuration. There is therefore no `Announce` message in the worst-case traversal time analysis. When multiple domains are used in an experiment, we set their spanning trees to the same configuration as the one pictured Figure 9.1. Using identical domains puts us in the worst-case scenario of link occupancy by synchronization messages (which is of course of little interest in terms of robustness). It should also be noted that this network reaches its maximum robustness with only two spanning trees per Grandmaster. However, our aim is to test the impact of up to 20 domains on the worst-case traversal time of other flows to get close to the situation of the AFDX configuration where 19 domains are required to avoid the multiple time base issue discussed in Chapter 7.

9.2 Impact of `Sync` frequency.

A reference Worst-Case Traversal Time (WCTT) measurement is carried out first with Timaeus-Net using a single domain with Grandmaster OBC A and default protocol configuration values, i.e. *syncInterval* of 125ms and *pdelayInterval* of 1s. Four other analyses were performed using *syncInterval* of 500ms, 250ms, 62.5ms and 31.25ms (resp. 2, 4, 16 and 32 messages par second). The results of these four analyses for seven flows are compared with the reference experiment in Figure 9.2. Only one flow per priority level is illustrated here, but the conclusions remain applicable to other flows of the same priority.

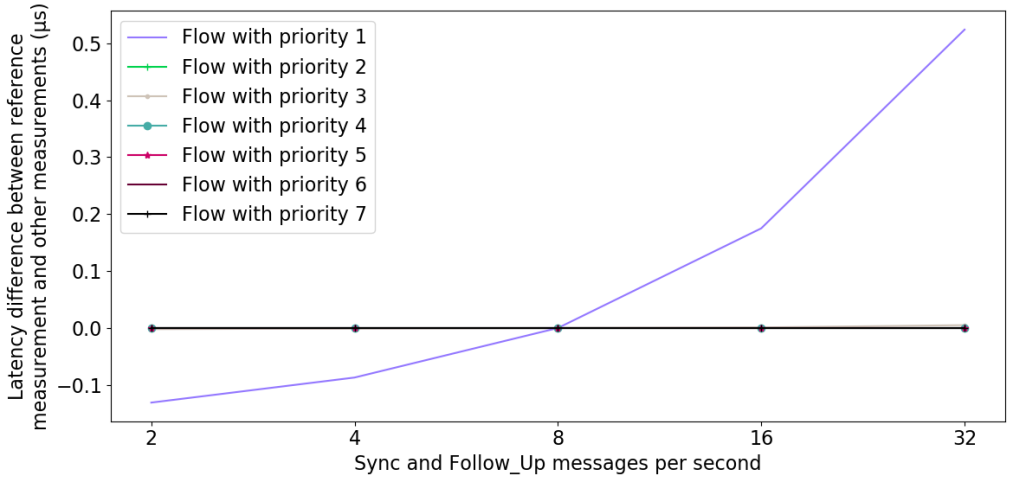


Figure 9.2: Impact of the number of `Sync` and `Follow_Up` messages per second on the worst-case traversal time for seven different priority flows. Synchronization flow is assigned priority 4.

Thanks to Figure 9.2, we note that there is a very limited impact on WCTT as a function of *syncInterval*. We observe a difference on certain lower priority flows. In addition, this impact is

very small in relation to the WCTT, with only a variation of between -130ns and +524ns compared to the reference experiment. In terms of comparison, the WCTT of the flow that undergoes these variations is around 300µs.

More specifically, we observe flows with a higher priority than synchronization, i.e. from 7 to 5, that are not at all affected by the synchronization frequency. For priority 7 flows, this is due to the time segregation performed by the Time-Aware Shaper. For priority 6 and 5 flows, the multiplication of synchronization messages is not an event that will induce worst-case conditions. Indeed, the worst-case residence time in the output FIFOs port for a priority 6 flow is reached when the largest of the lower-priority frames just starts to be sent when the priority 6 frame arrives in the FIFO. The latter will then have to wait until the end of its transmission before being sent. For a priority 5 frame, the worst-case is reached when you have to wait for the largest of the lower-priority frames to finish sending, and then for all the priority 6 packets in the FIFO to be sent. However, synchronization frames are far from being the largest frames on the network, with less than 100 bytes, compared with the 1500 bytes of the largest frames in this case study.

The same observation can be made for flows with the same priority than synchronization messages, i.e. priority 4. However, this time this result is explained by the very low amplitude of change, which makes the result non-observable. This explanation is confirmed by Figure 9.3 which illustrates the results for the same experiment when 20 domains run in parallel. We can see a variation of between -1ns and +6ns around the reference WCTT obtained at 125ms with 20 domains for priority 4 flow.

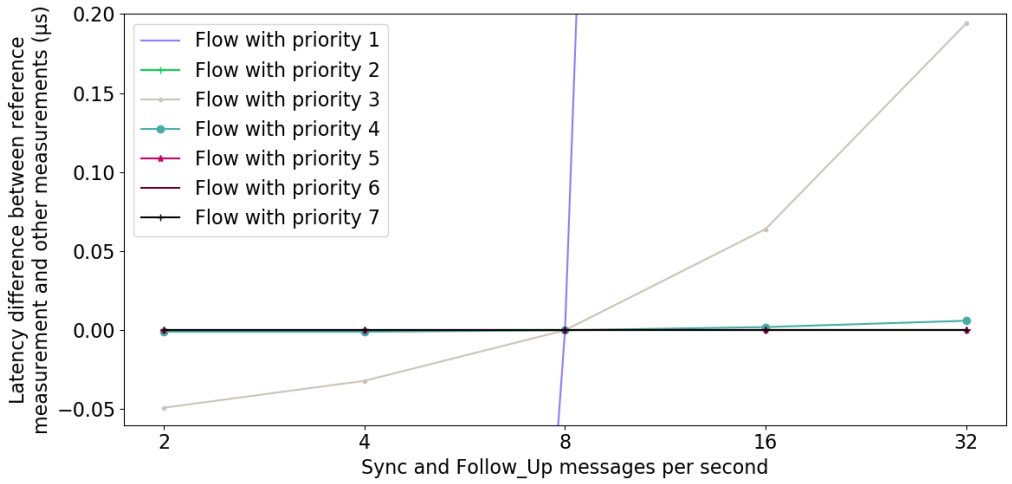


Figure 9.3: Impact of number of `Sync` and `Follow_Up` message per second on worst-case traversal time for seven different priority flows with 20 domains. A zoom is used to highlight the WCTT evolution of the priority 4 flow.

For lower-priority flows, we observe a variation in WCTT for priority 1 flow and very little variation for priority 3 flow. For priority 1, the analysis tool computes a variation of between -130ns and +524ns when compared with the 1-domain reference experiment, and a variation of between -2.632µs and +10.538µs when compared with the 20-domain reference experiment. For priority 3, the variation is between -2ns and +4ns compared with the 1-domain reference experiment and between -50ns and +20ns compared with the 20-domain reference experiment.

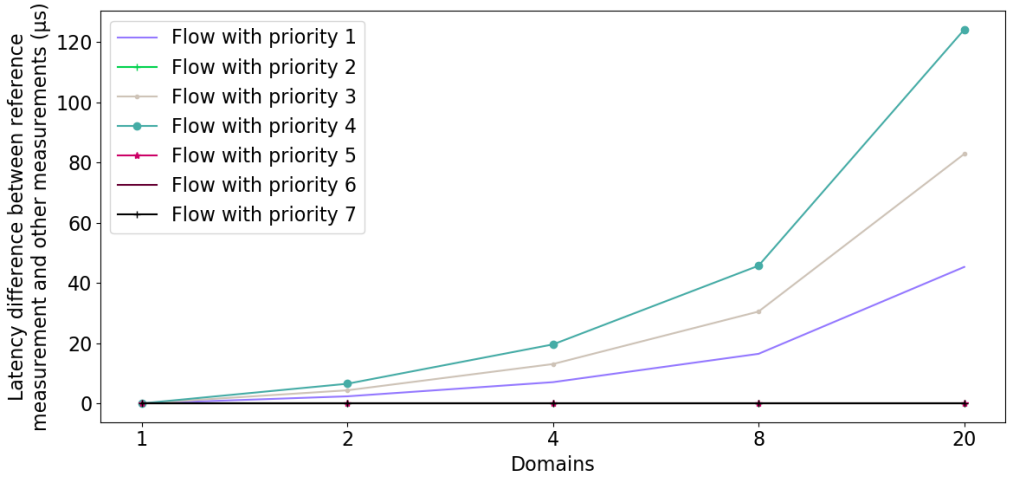


Figure 9.4: Impact of number of domains on worst-case traversal time for seven different priority flows.

The priority 1 flow is much more affected than the priority 3 flow, due to its data volume and path through the network. Indeed, this flow has crossed the two busiest links in the network, with its path going from Instrument 1 to the Solid State Mass Memory (SSMM), the central memory of the satellite. Instrument 1 is the device that produces the largest amount of data, and the SSMM is the network's biggest data consumer. The priority 1 flow is also a data-intensive flow, and must therefore be fragmented into several Ethernet frames. As the synchronization frequency increases, these bursts of multiple Ethernet frames may be delayed several times due to synchronization messages in the same output port, in contrast to flows carrying fewer data, such as priority 3 flow.

Furthermore, during these two series of analyses, the priority 2 flow was never impacted by the frequencies of `Sync` and `Follow_Up` messages. An analysis of the path taken by this flow shows that it never shares any output port with the `Sync` and `Follow_Up` messages. In fact, this flow takes the opposite path to `Sync` and `Follow_Up`, which are sent by OBC A to all devices, whereas the priority 2 flow is sent by NAVCAM to OBC A. This lack of influence of *syncInterval* due to a path without `Sync` and `Follow_Up` can also be observed on other priority 1, 3 and 4 flows.

9.3 Impact of the number of domains

As in the previous section, a reference analysis of WCTTs was performed with a single domain and the default gPTP parameter of 0.125s *syncInterval* and 1s *pdelayInterval*. This reference analysis was then compared with two series of four analyses exploring the use of 2, 4, 8 and 20 synchronization domains. The two series differ in the use or absence of the Common Mean Link Delay Service (CMLDS) mechanism. As a reminder, when activated, this mechanism enables the propagation delay measurement mechanism to be executed in a single domain and the result passed on to the other domains, thus reducing the number of *Pdelay* messages exchanged. The results obtained are described in Figure 9.4 and Figure 9.5.

These two figures show that the impact is greater than the one of the *syncInterval*, with an increase in WCTT of up to 124µs with CMLDS and 239µs without. For the concerned flow, this

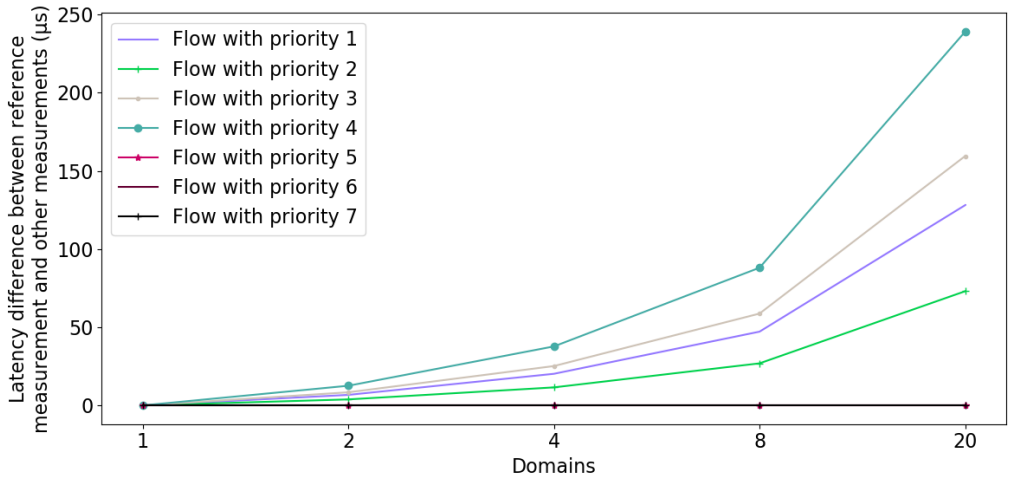


Figure 9.5: Impact of number of domains on worst-case traversal time for seven different priority flows when CMLDS is disabled.

represents a twofold increase in WCTT with CMLDS and a threefold increase without CMLDS, compared with the reference experiment.

More specifically, we can see that the highest priority flows (5, 6 and 7) are not affected, for the same reasons as in the previous *syncInterval* experiment.

This time, the flow with the same priority as the synchronization messages is the one the most affected of the 7 flows, with an increase in its WCTT of 124µs with CMLDS and 239µs without CMLDS. Among Priority 4 flows, this is also the one with the greatest increase in WCTT. It's also the only flow at this priority level to make three hops, compared with two for the other priority 4 flows. These other priority 4 flows experience an increase in WCTT of up to 82µs with CMLDS and 160µs without CMLDS with 20 domains. This means an increase of 42µs with CMLDS and 80µs with CMLDS per hop. Therefore, the flow studied passes through one more output port and, in the worst-case, is delayed once again by all the `Sync` and `Follow_Up` messages from all the domains.

For lower-priority flows, there is a noticeable increase in WCTT as the number of domains increases for priority 1 and 3 flows. Indeed, in the worst-case, `Sync` or `Follow_Up` messages from each domain, end up filling the FIFO 4 first, and therefore delay the transmission of frames of lower priority or even same priority than synchronization frames. Furthermore, as for the previous section, there is no impact for priority 2 flows when CMLDS is used. This can also be explained by the path taken by messages in priority 2 flows. On this path, messages never compete with synchronization messages in the FIFOs. On the other hand, when CMLDS is deactivated, the increase in the number of *Pdelay* messages affects the WCTT of the priority 2 flow. We also note that priority 4 is more affected than the other three lower-priority flows. As previously, the priority 4 flow includes three hops, compared with two for the others. In the worst-case, therefore, the lower-priority flows are delayed one less time by the *Pdelay* messages in the output ports than the messages in the priority 4 flow.

Compared with the previous experiment with *syncInterval*, we can see that increasing the number of domains has a much greater impact on the WCTT than increasing the number of synchronization messages per second. For example, for the Priority 1 flow with CMLDS, going from a *syncInterval*

of 125ms to 62.5ms leads to an increase in WCTT of 0.175µs, whereas going from one domain to two leads to an increase in WCTT of 2.350µs. Although the average bandwidth consumed by these two example configurations is identical, the impact on WCTT is more than 13 times greater in the case of increasing the number of domains. Indeed, increasing the frequency of synchronization messages simply reduces the sending period, but does not create the conditions for several `Sync` or `Follow_Up` messages to end up in the FIFO at the same time. Whereas with multiple domains, in the worst-case, `Sync` or `Follow_Up` messages from all domains inducing a longer waiting time in the same or lower priority FIFOs than synchronization frames.

9.4 Conclusion

From our experiments, we can note that the synchronization frequency has a slight influence on the WCTT of flows with the same priority or with a lower priority than synchronization. In the case of the satellite case study, the impact on flows of the same priority is less than a nanosecond with 1 domain, and becomes barely observable when using 20 domains, with a variation of less than 10ns compared with the reference experiment. Lower-priority flows are impacted by up to an additional 524ns for a single-domain configuration when the *syncInterval* is reduced from 125ms to 31.25ms, i.e. an increase in WCTT of 0.2%. The impact is also linked to the path taken by the flows studied.

With multiple domains, the increase in WCTT is greater than with *syncInterval*. This increase is due to the `Sync` and `Follow_Up` messages from multiple domains that may fill the FIFO used for synchronization at the same time in worst-case. We also observe few tens of microseconds variation of the WCTT increase due to the distance the flow travels. In addition, there is a difference of up to 115µs in WCTT when CMLDS is used or not used. This observation shows the importance of this simple mechanism on the WCTT of flows with which gPTP shares the network. However, it should be kept in mind that 20 domains with an identical spanning tree on this topology is a very pessimistic hypothesis. In practice, this topology should host 4 spanning trees covering the entire network. Two spanning trees would be rooted at OBC A and two at OBC B. The spanning trees of the same Grandmaster would be independent. The greatest number of spanning trees on the same link would then be two spanning trees, each from a different GM, when both OBCs are switched on, despite cold redundancy. With this more realistic hypothesis, the difference in WCTT between the reference experience and a configuration with two on the same link is only 6.5µs at most.

All these impacts are non-existent on flows of higher priority than synchronization messages. They are more significant on flows of the same or lower priority, which are often much less constrained. Moreover, these impacts are predictable and can be computed using worst-case latency analysis tools in the network design phase.

This chapter also shows the importance of reducing the priority of synchronization messages to meet the latency constraints of the most constrained flows, despite the standard's recommendations to consider synchronization messages as having the highest priority. In the event that synchronization messages are the cause of a WCTT constraint overflow, it is possible to reduce the priority of synchronization messages. This reduction will have an impact on synchronization precision, as the `Sync` and `Follow_Up` messages will be delayed, but this can be quantified in the worst-case using the J_{fup} parameter of the model presented in Chapter 5.

Part V

Conclusion

Chapter 10

Conclusions and perspectives

10.1 Conclusions

In this manuscript, we aimed at mitigating the obstacles blocking the deployment of IEEE802.1AS in critical embedded networks. Therefore, our objective was to propose the keys to achieving precise and robust synchronization for such networks. The work was divided into three parts. The first part consisted in studying the precision achievable with the synchronization protocol. The second part studied the robustness mechanisms proposed by the standard and their configurations. And finally, the third part concludes our work by putting the previous results into practice on a case study of a unified network for satellite, in order to validate the protocol's low impact on other data flows. The main takeaway results of these three parts are given next.

The state-of-the-art study of IEEE802.1AS precision has highlighted three ways of studying it: simulation, experimental measurements and formal methods. However, work based on simulation or the use of formal methods lacks comparative analysis with hardware. Thus, in the first chapter of Part II, we improve an open source gPTP simulation library by introducing realistic sources of inaccuracy such as clock granularity, physical layer jitter and asymmetries introduced characteristic of a given physical layer technology. After making these changes to the simulator, an inaccuracies source calibration phase is carried out using switches supporting IEEE802.1AS. This calibration phase is reproducible for different time-aware systems and/or physical layers, and requires no special equipment other than an oscilloscope or PPS analyser. In our case, we calibrate the simulator for Fraunhofer IPMS FPGA switches with 100Mb/s and 1Gb/s copper twisted-pair links. After calibration, RMSEs of approximately 3ns between sliding average of the precision measured and simulated one is achieved, confirming the high representativeness of the simulator. This enhanced library is available open source at the following URL: `https://github.com/irit-rmess/gPtp-implementation`

Even though simulation is extremely useful for testing new mechanisms without having to wait for a hardware implementation, or for rapidly exploring the huge range of possible configurations, it is not a tool that can provide the necessary guarantees for a critical system. Indeed, in this field, it is common to turn to formal methods to provide a proof of deterministic worst-case operations. Building on the knowledge of realistic inaccuracies sources gained from our work on the simulator, we have proposed improvements to a worst-case precision bound model in the second chapter of Part II. The idea of this model is to make an inventory of all the inaccuracies that can impact each measurement made by the protocol. Using this inventory, it is possible to compute the maximum error caused by the different measurements steps of the protocol. We compute the largest achievable error, which when combined with the worst-case clock drift determines a bound on precision. Even

with the addition of new sources of inaccuracies, such as the asymmetries caused by physical layers, our finer modelling of existing sources leads us to reduce error bound by two compared to the literature model.

Besides its tighter modeling power, our model has the following original features:

- a generic physical layer model. This layer can be configured using physical layer characteristics. It avoids the update and derivation of specific equations in the global bound model.
- a lower bound on the worst-case precision that holds in case of a negative drift.
- a thorough validation using experimental measurements and simulations: an exhaustive worst-case search highlights a good hardware representativeness, achieving a very low level of pessimism. To be more specific, a pessimism below 10% is achieved with a 100Mb/s copper physical layer, and below 21% with 1Gb/s.

Our model makes it possible to guarantee a bound on worst-case precision, offering distributed applications and Time-Aware Shaper the possibility to be calibrated according to a bound on the synchronization protocol precision.

In terms of robustness to faults, which is discussed in Part III, the state of the art highlights a cruel lack of work on the static port state configuration mechanism combined with the use of multiple domains. However, in the world of critical embedded systems, such mechanisms are generally preferred for reasons of simplicity, leading to determinism that is easier to prove. As a first step, we carried out a comparative study between the two robustness mechanisms proposed by the standard, i.e. the Best Master Clock Algorithm and the static configuration of the port state with multiple domains. This study, that relies on five qualitative and quantitative metrics, shows that the performance of both mechanisms in terms of mitigation power and reconfiguration time is relatively similar for the networks of 3 and 4 switches considered here. However, the predictability of BMCA reconfiguration time on a more complex network remains to be proven. And the biggest obstacle in using the static mechanism is complexity of choosing a configuration at design stage.

In the second chapter of Part III, a method for finding a robust and precise multi-domain static configuration and Grandmaster is proposed. Indeed, the difficulty with this mechanism is to find a configuration that suits the need among the thousands of possible configurations. We have therefore proposed a method for finding the optimum configuration to meet the needs of our industrial case studies, i.e. the most precise configuration among the most robust. Our method is based on characteristic evaluation using two metrics: one for precision and one for robustness to failure. However, our method is based on brute force evaluation of numerous possible configurations. It is therefore particularly well-suited to the small networks of the embedded world, and finds the solution in conceivable time for a configuration calculated offline, as shown by the results of the case studies in this manuscript. For larger and more connected networks, such as factory automation networks, this method is not suitable, as the number of possible configurations explodes. We have also shown that the two metrics can be diverted from their primary use in order to optimize Grandmaster's placement and topology in terms of failure robustness and/or precision.

And finally, in Part IV, we apply the results of the previous two parts to the satellite use case in order to study the impact of synchronization on TAS configuration and on other traffic present in the network. As far as TAS configuration is concerned, we have shown that the bandwidth loss due to the guard band added to the TAS window is very low. In addition, we highlighted the existence of a minimum wasted bandwidth. This result also enabled us to show that the need for precise synchronization below the microsecond is too demanding for most Time-Aware Shaper applications.

We have also illustrated in an example the use of our results to select TSN switches that can meet requirements related to a given minimum usable bandwidth.

In a second step, we took the opposite path to the literature, which studies the impact of network load on precision, by investigating the impact of the synchronization protocol and its configuration on the Worst-Case Traversal Time (WCTT) of other network flows. Indeed, these flows must respect latency constraints for the proper functioning of the distributed applications that consume them. To understand the impact of synchronization frequency, which is a parameter that determines precision, and of the number of domains, which is a parameter that determines robustness to failure, analyses were carried out using network calculus. These analyses highlighted the very weak impact of *syncInterval* on WCTTs, and the more important but still weak impact of the number of domains. Indirectly, this chapter also illustrates the importance of reducing the priority of synchronization messages so as not to impact the most constrained flows. These analyses also highlighted the importance of using the Common Mean Link Delay Service to greatly reduce the impact of synchronization on WCTTs.

10.2 Perspectives

We have seen that IEEE802.1AS can meet embedded needs in terms of precision and robustness to failure, while having a low impact on other network traffic. However, we believe that the following topics still need to be explored before the protocol's capabilities can be fully utilized in a critical embedded context. This perspective section is organized like the three parts of the manuscript.

10.2.1 Precision

A better understanding of the different combinations of jitter impacting the timestamping, and especially the jitter between the physical layer and the clock, could help to increase the representativeness of the simulation and reduce the pessimism of the bounds.

A probabilistic bound model on the worst-case precision could also be an interesting addition for less critical embedded systems, such as the case of digital audio in the aircraft cabin.

The time required to establish synchronization is also a point that needs to be studied in fast start networks such as automotive networks. This duration could be reduced by using `Signalling` messages to request a high frequency of synchronization at start-up, then reducing it once synchronization has been established.

This establishment time should also be studied in all types of network when the Time-Aware Shaper is used. Indeed, using a bandwidth time-sharing mechanism when synchronization is not yet established risks slowing down the establishment of synchronization due to the desynchronization of the various network devices, especially if the transmission opportunities offered to gPTP are of short duration.

Although outside the scope of the standard, the clock servo is an interesting feature. The clock servo is the algorithm that uses the offset from the GM time calculated by gPTP to slowly or rapidly correct the clock according to given needs. Application-specific clock servos already exist, for example on satellites where synchronization is transmitted via PPS links. However, it would be interesting to find a servo clock ideal for Time-Aware Shaper or Hot Standby Grandmaster or other synchronous TSN mechanisms to implement in network devices.

10.2.2 Robustness

The biggest brake on the adoption of 802.1AS in the critical environment is robustness. Indeed, this point, because of the static mechanisms, is still being standardized in P802.1ASdm. It will be necessary to continue the work on this subject when the standard is published. The first step will be to make sure that the hypothesis we have made are correct. If not, a new evaluation of the difference between static and dynamic configuration will be necessary. Then, the case of failure that we did not treat in the manuscript should be studied, like the one where a Grandmaster shares a corrupted time base.

For less critical embedded networks, a bound on BMCA reconfiguration time would be an interesting work to enable simple and very robust configuration.

The optimisation of the topology, or more generally of the network, to make synchronisation more robust and/or precise is also a point that remains to be explored.

10.2.3 Synchronization and the other network activities

Moreover, regarding the impact of synchronization on network traffic, we have studied the TAS and the impact of the number of 802.1AS messages, but we have not treated the TSN 802.1Qci mechanism. The latter is a policing protocol that aims at monitoring that flows respect their emission contract. To do so, this mechanism can use asynchronous flow counter or synchronous mechanism allowing to check the sending of a message during a precise time window. As we have done for the TAS, the relationship between the precision and this mechanism should be studied.

Part VI

Appendices

Appendix A

Résumé Long Français

Cette annexe résume le manuscrit en français. Les développements ne seront que succinctement expliqués afin de concentrer le résumé sur l'analyse des résultats.

A.1 Introduction

L'objectif de la thèse est de répondre au besoin de synchronisation à la fois robuste et précise avec IEEE802.1AS dans les réseaux embarqués critiques TSN. Dans cette première section, le contexte dans lequel se déroule cette thèse ainsi que TSN et IEEE802.1AS seront définis. Cette section se conclura sur un résumé de l'état de l'art afin d'expliciter la direction prise dans cette thèse.

A.1.1 Contexte

Cette thèse traite de réseaux embarqués critiques. Un réseau est appelé réseau embarqué critique lorsque qu'il interconnecte des systèmes embarqués et critiques. Les systèmes embarqués sont des systèmes informatiques et électroniques autonomes et souvent temps-réel qui exécutent une tâche au sein de l'appareil auquel ils sont intégrés. Ils sont limités en ressource comme la puissance de calcul, la mémoire, la consommation énergétique, le poids, l'encombrements. Pour ce qui est de l'aspect criticité, ce dernier est lié à l'impact d'un dysfonctionnement qui peut aller jusqu'à causer la perte de nombreuses vies.

On trouve des réseaux embarqués critiques dans différents domaines. Dans ce manuscrit, nous nous concentrerons seulement sur les réseaux des secteurs de l'aéronautique, de l'automobile et du spatial.

Historiquement et dans les secteurs d'intérêt pour la thèse, la communication entre les différents systèmes embarqués critiques a d'abord reposé sur l'utilisation de liens dédiés. Pour répondre à l'augmentation du nombre de ses systèmes, des bus déterministes ont été proposés pour répondre à différent besoin comme CAN [10], LIN [28] et MIL-STD-1553 [1]. Ces bus permettent le partage d'un même lien par différents systèmes, réduisant ainsi le nombre de liens nécessaire. Cependant, ces bus avec leur quelques Mb/s au mieux ne permettent pas de répondre au besoin des nouvelles applications très gourmandes en bande passante. Ainsi, des réseaux déterministes et à plus grand débit ont été proposés comme l'AFDX [15], Time-Triggered Ethernet [56] et SpaceWire [17]. Bien qu'utilisées dans les secteurs qui ont motivé la création de ces nouveaux réseaux, ses solutions souffrent de leur nature propriétaire et de leur faible demande, les rendant ainsi très couteuses. C'est dans ce contexte que l'IEEE propose une nouvelle solution réseau appelé Time-Sensitive Networking qui est détaillée dans la sous-section suivante.

A.1.2 Time-Sensitive Networking

Time-Sentive Networking (TSN) est un groupe de travail de IEEE qui standardise un ensemble de standards, couramment nommés standards TSN. Ces standards proposent des protocoles visant à rendre Ethernet déterministe et permettre l'envoi de données sensibles au temps. Historiquement dédié à l'audio/vidéo, sous le nom standard Audio Video Bridging (AVB), ils attirent aujourd'hui l'intérêt de nombreux secteurs. Parmi ces secteurs, on retrouve l'automobile, l'aéronautique et le spatial.

Pour atteindre ces objectifs, le groupe de travail TSN a défini de nombreux standards. Ces standards peuvent être divisés en quatre groupes en fonction de leur objectif comme représenté sur la Figure A.1. On y trouve ainsi des standards dédiés à la synchronisation des appareils du réseau, au controle de la latence des trames, à la fiabilisation des échanges et à la gestion des ressources du réseau.

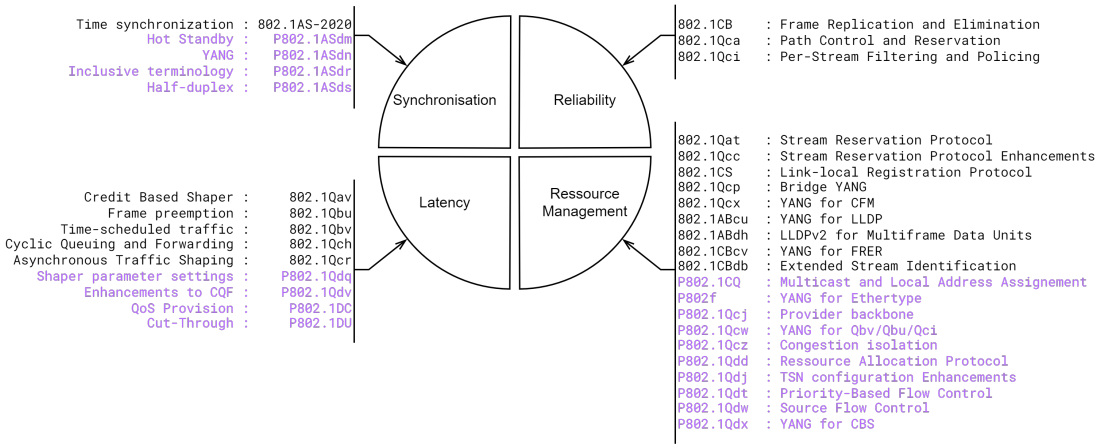


Figure A.1: Standards TSN par groupe. Les standards en cours de définitions sont représentés en violet

En rendant Ethernet déterministe, TSN pourrait permettre aux réseaux du monde de l'embarqué de profiter des avantages d'Ethernet tel que le grand choix de couches physiques permettant ainsi de répondre au besoin allant de quelques Mb/s à plusieurs Gb/s, le cout faible des composants Ethernet, mais aussi la possibilité d'utiliser le réseau avec des messages de différents niveaux de criticité, grâce aux huit niveaux de priorités Ethernet.

A.1.3 IEEE802.1AS

Parmi les mécanismes proposés par le groupe de travail TSN, on retrouve plusieurs mécanismes basés sur le temps comme le Time-Aware Shaper (TAS) qui permet de contrôler en temps l'accès au medium. Pour un fonctionnement optimal, ces mécanismes nécessitent une base de temps commune dans l'ensemble du réseau. Une telle base de temps peut aussi profiter aux applications distribuées. Pour obtenir cette base de temps commune, le groupe de travail a proposé IEEE802.1AS [34].

IEEE802.1AS se base sur un protocole déjà existant, appelé IEEE1588 [32], et l'étend pour une utilisation avec les restes de mécanismes TSN. L'implémentation de IEEE802.1AS est nommé generalized Precision Time Protocol (gPTP) par extension de IEEE1588 et Precision Time Protocol (PTP). gPTP se différencie des protocoles existant par son objectif d'atteindre une précision de synchronisation sous la microseconde dans un réseau à sept sauts avec du matériel commun et peu couteux.

Les protocoles de synchronisation reposent sur deux mécanismes principaux. Le premier est l'estimation du temps entre l'instant d'envoi de l'information de synchronisation et l'instant du traitement. Le deuxième est la distribution des informations de synchronisation.

Afin d'obtenir une synchronisation précise, la première étape est de calculer le délai subit par le message de synchronisation entre son envoi et son traitement pour le prendre en compte lors du calcul de la correction à appliquer. Pour ce faire, IEEE802.1AS propose le mécanisme appelé *peer-to-peer propagation delay measurement* qui mesure le délai de propagation en point à point. Ce mécanisme est exécuté périodiquement et par tous les ports gPTP. Il repose sur l'envoi de trois messages qui vont générer quatre horodatages nécessaires pour les calculs. L'échange qui repose sur

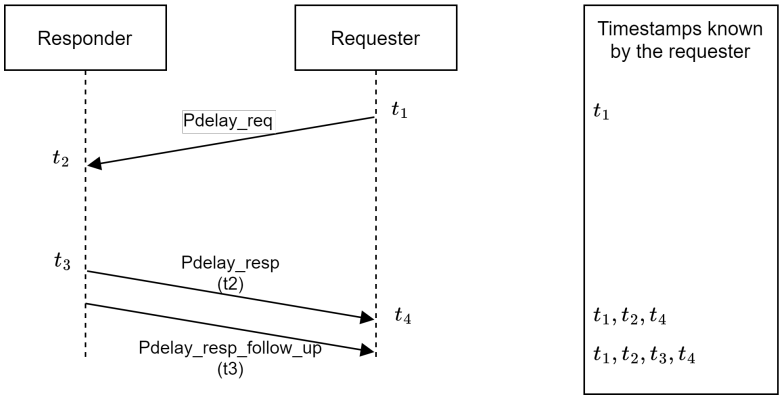


Figure A.2: Illustration des échanges effectués par le mecansime peer-to-peer propagation delay measurement

les messages `Pdelay_Req`, `Pdelay_Resp` et `Pdelay_Resp_Follow_Up` est décrit sur la Figure A.2.

À partir de ces quatre horodatages, le protocole estime le délai de propagation D entre le demandeur et le répondeur à l'aide de la formule suivante :

$$D = \frac{(t_2 - t_3) + nr(t_4 - t_1)}{2} \tag{A.1}$$

nr est appelé *neighborRateRatio* et permet de prendre en compte l'écart de fréquence entre l'horloge des deux systèmes. Il est calculé à l'aide de l'équation suivante :

$$nr = \frac{f_{req}}{f_{resp}} = \frac{t'_3 - t_3}{t'_4 - t_4} \tag{A.2}$$

Pour la distribution des informations de synchronisation, comme la plupart des protocoles de synchronisation actuels, gPTP repose sur une distribution hiérarchique des informations temporelle. Un équipement de référence, appelé Grandmaster, distribue, périodiquement, sa base de temps à l'ensemble des appareils supportant le protocole à l'aide d'un arbre couvrant, comme décrit sur la Figure A.3. Cet arbre couvrant est le résultat de l'état du port. Le protocole propose trois états : Maitre, Esclave, Passif. En pratique, le Grandmaster émet périodiquement des messages `Sync` et `Follow_Up` sur ses ports Maitre (M sur la Figure A.3). Chaque appareil supportant gPTP recevant ces messages sur son port Esclave (S sur la Figure A.3) les transfèrent à ses ports Maitre, s'il en possède, afin d'être émis vers les autres appareils. Les ports passifs (P sur la Figure A.3) les ignorent pour éviter les dépendances cycliques.

Les messages `Sync` agissent comme un déclencheur pour l'horodatage de plusieurs événements nécessaire pour le calcul de l'écart entre l'horloge en question et celle du Grandmaster comme illustré dans la Figure A.4. Le premier de ces horodatages est l'envoi du `Sync` par le Grandmaster. Il est noté O et est transporté par les messages `Follow_Up`. Les autres événements sont la réception et l'émission du message `Sync` par les différents systèmes qu'il traverse. Avant chaque envoi du message `Follow_Up`, celui-ci est mis à jour avec *rateRatio* r et *correctionField* C fraichement calculé par le protocole.

Le *rateRatio* joue le même rôle que le *neighborRateRatio* mais pour l'écart de fréquence entre le Grandmaster et l'appareil qui le calcule. Il est initialisé à 1 par le premier et calculé par le deuxième à l'aide de l'équation suivante.

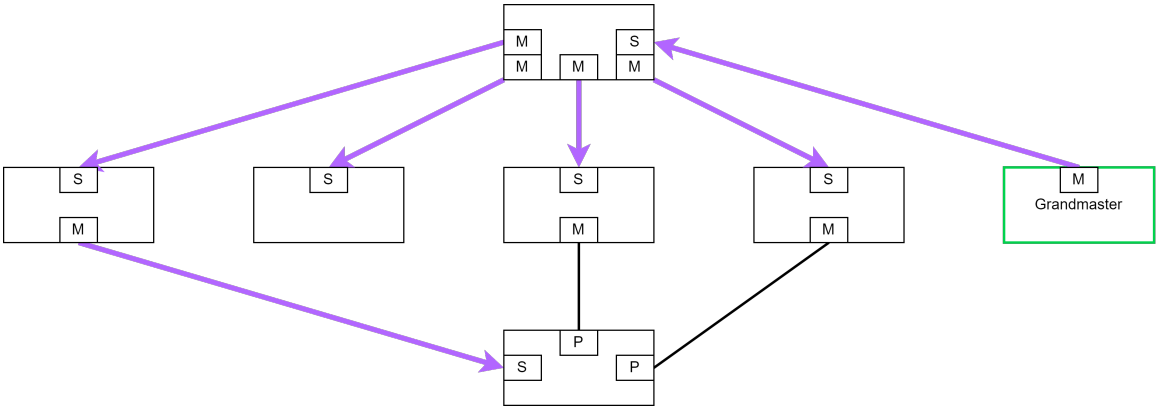


Figure A.3: Illustration d'un des arbres couvrants possibles pour distribuer les informations de synchronisation sur la topologie de cœur automobile

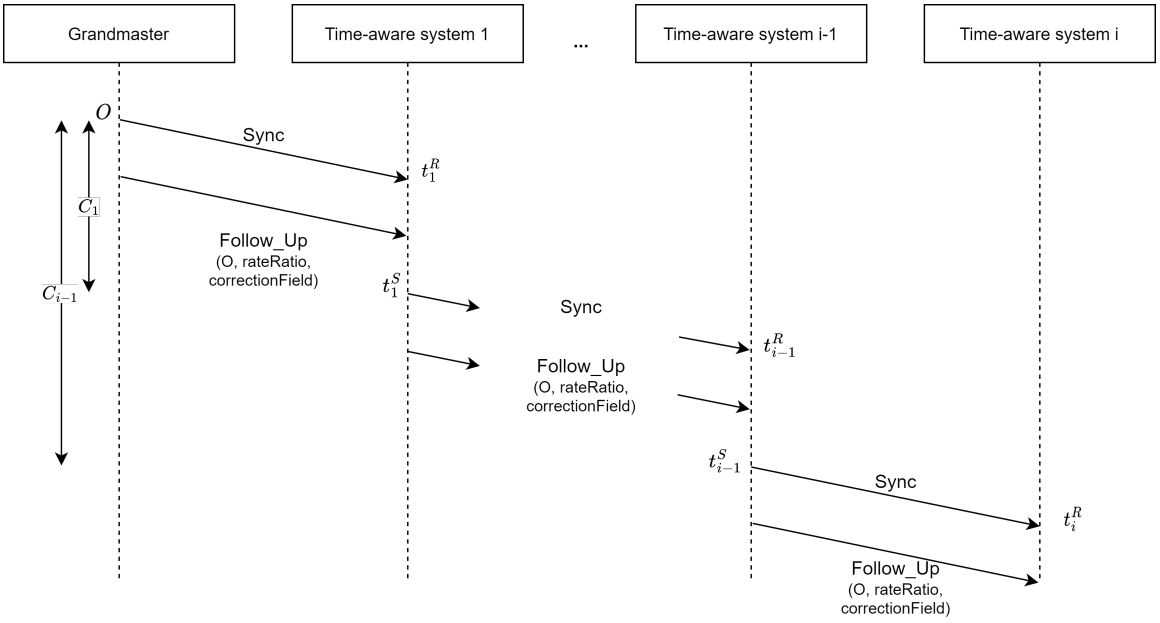


Figure A.4: Illustration du mécanisme de distribution des informations de synchronisation dans un réseau à i sauts

$$r_i = r_{i-1} \times nr_i \tag{A.3}$$

Le *correctionField* C transport le temps qui s'est écoulé depuis l'envoi du `Sync` par le Grandmaster jusqu'à l'avant-dernier saut. Au saut i, C_i est calculé à l'aide du précédent *correctionField* C_{i-1}, de la valeur actuel de D_i et du temps de résidence du `Sync` dans l'appareil i $t_i^S - t_i^R$ en utilisant l'équation suivante.

$$C_i = C_{i-1} + D_i \times r_{i-1} + (t_i^S - t_i^R) \times r_{i-1} \times nr_i \tag{A.4}$$

À chaque réception de `Sync + Follow_Up`, l'appareil i calcule son décalage avec l'horloge du Grandmaster afin d'en déduire la correction à appliquer à son horloge. Pour ce faire, il compare son horloge à l'horloge estimée du Grandmaster $GM_i(t)$. $GM_i(t)$ est calculé à l'aide de l'équation suivante.

$$GM_i(t) = O + C_{i-1} + D_i + (t - t_i^R) \tag{A.5}$$

Où O est l'horodatage d'émission du `Sync` par le Grandmaster, C_{i-1} le *correctionField* du message `Follow_Up`, D_i le délai de propagation estimé par le mécanisme *peer-to-peer propagation delay measurement* entre l'appareil i et l'appareil précédent $i-1$ et $(t - t_i^R)$ le temps écoulé depuis la réception du `Sync`.

En plus de ses mécanismes principaux, le standard propose quelques mécanismes additionnels. Dans ce résume, seul ceux abordés dans la suite seront brièvement présentés ci-dessous.

Le standard propose deux mécanismes pour déterminer l'état des ports. Le premier est un mécanisme dynamique appelé Best Master Clock Algorithm (BMCA). Le BMCA est un algorithme distribué de convergence utilisant des messages appellé `Announce` et émit périodiquement par les potentiels Grandmasters à destination de l'ensemble du réseau afin de déterminer l'arbre couvrant optimal. Le deuxième mécanisme est appelé External Port Configuration. Derrière ce nom se cache un simple mécanisme de configuration statique de l'état de chaque port.

IEEE802.1AS propose aussi un mécanisme de domaine. Un domaine est un ensemble d'appareil partageant la même base de temps ainsi que le même arbre couvrant utilisé pour la distribution des messages `Sync`. Pour effectuer la différentiation, l'entête des messages gPTP contient l'identifiant du domaine. La Figure A.5 illustre les possibilité de ce mécanisme. On y observe ainsi la possibilité de partager la base de temps de deux Grandmaster différent sur le même réseau. Ces deux bases de temps peuvent être proches ou complètement différentes, par exemple une base de temps UTC et une base de temps de travail initialisé lors du démarrage des appareils. L'autre possibilité offerte par ce mécanisme est l'utilisation de plusieurs domaines ayant le même Grandmaster. Ceci permet d'augmenter la disponibilité du système en cas de panne grâce à des arbres couvrant redondant.

A.1.4 Revue de littérature

Les travaux étudiés lors de la revue de littérature autour du standard IEEE802.1AS peuvent être regroupés en deux catégories : précision et robustesse.

Les travaux autour de la précision mettent en avant trois méthodes d'étude de la précision : la simulation, l'expérimentation et les méthodes formelles. Pour ce qui est de la simulation, on trouve de multiples travaux, [61] [72] [62] [51] décrivant une implémentation d'une librairie de simulation qui est utilisé à des fins variables comme la vérification de performance ou bien l'étude de l'impact du trafic réseau sur la précision. Deux librairies ce démarquent grâce à leur aspect open-source [81] et [90]. À l'exception de la librairie de [62], les librairies de simulation n'implémentent pas de source d'imprécision autre que la dérive de l'horloge ainsi que parfois sa granularité. Or les travaux [74]

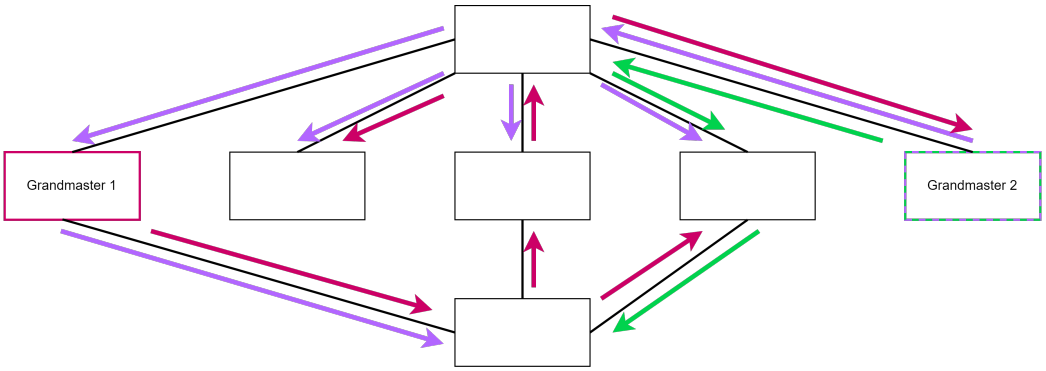


Figure A.5: Illustration des possibilités offertes par les domaines avec une topologie automobile simplifiée

ont montré l'existence et l'impact d'autre source liée à la couche physique utilisé. De plus, aucune de ces librairies n'a comparé ces résultats avec des implémentations matérielles du protocole.

Pour ce qui est de l'aspect expérimental, on retrouve aussi plusieurs travaux qui ont étudié l'impact de paramètre dimensionnant de la précision. Comme les travaux de [83] qui montre que la précision atteignable peu atteindre 25 ns après trois sauts avec une implémentation sur FPGA du protocole. [66] et [61] atteignent quant à eux des précisions de l'ordre de 500 ns après sept sauts. [63] a montré que le trafic à un impact négligeable sur la précision lorsque des mécanismes de qualité de service sont utilisés pour rendre les messages de synchronisation prioritaire. Et comme mentionné brièvement précédemment, [74] étudie expérimentalement l'impact de la couche physique sur la précision. L'ensemble de ces travaux met en avant plusieurs paramètres impactant la précision, cependant l'utilisation de matériels différents pour chaque travail rend impossible une comparaison fine de ces résultats.

Pour une utilisation dans un environnement critique tel que ceux étudiés dans ce manuscrit, des preuves formelles sont régulièrement utilisées pour prouver le bon fonctionnement d'un système dans toutes les conditions possibles. Seuls deux travaux ont étudié le protocole sous cet angle. [38] utilise UPPAAL pour prouver le bon fonctionnement du protocole. Cependant, la borne sur la précision obtenue avec ces travaux n'est pas représentative de la réalité à cause du manque d'implémentation de source d'imprécision réaliste. Contrairement aux travaux de [62] qui implémente ces sources d'implémentation réaliste pour la couche physique 100Base-T à l'aide des travaux de [74]. Mais comme les travaux de simulation, ces travaux souffrent d'un manque de comparaison avec des expérimentations afin de valider les modèles proposés.

La deuxième catégorie de travaux traite de l'aspect robustesse. Cette deuxième catégorie peut elle-même être divisée en deux sous-catégories. La première catégorie porte sur la robustesse aux pannes. En effet, les pannes de lien ou d'équipement sont inévitables. Cependant, dans un réseau critique, elles doivent être mitigées avec un impact contrôlé sur le reste du réseau. [80] et [68] ont étudié la possibilité d'utiliser plusieurs arbres couvrants déterminés par le BMCA pour améliorer la disponibilité de la synchronisation. Cependant, au meilleur de notre connaissance, aucun travaux similaire n'a été effectué dans le cadre d'une configuration statique à plusieurs domaines. En effet, grâce à sa simplicité, cette solution pourrait être plus adaptée aux réseaux embarqués critiques. [57] et [49] ont étudié les cas de pannes où un Grandmaster envoi des informations erronées. Ils proposent de détecter les panne à l'aide d'estimateur statique ou d'utiliser les Grandmasters de secours comme surveillant. Simulation et expérimentation confirment l'efficacité de ces solutions.

La deuxième sous-catégorie regroupe les travaux autour de la robustesse aux attaques malveillantes. D'un point de vue sécurité, un protocole de synchronisation est une cible très intéressante de par son utilisation centrale par d'autres mécanismes TSN ou bien les applications distribuées. [60], [88] and [87] ont montré que les attaques suivantes était possible avec PTP :

- Grandmaster rebelle : Un attaquant prétend est un meilleur Grandmaster que le Grandmaster actuel pour être élue par le BMCA et prendre le contrôle du temps dans le réseau.
- Manipulation de paquet : Un attaquant en position d'homme du milieu falsifie les données de paquet légitime.
- Manipulation du délai des paquets : Un attaquant en position d'homme du milieu retarde les paquets légitimes.

Avec gPTP, [58] a montré qu'il était possible de réaliser des attaques par usurpation. Afin de limiter ces attaques, ils recommandent en autre d'utiliser IEEE 802.1AE [31] pour chiffrer la charge utile de trame MAC et plusieurs chemins a des fins de détections.

A.1.5 Problématique

Pour rappel, notre objectif est de proposer une solution de synchronisation robuste et précise pour répondre aux besoins des réseaux TSN embarqués critiques. Nous avons vu que le protocole IEEE802.1AS est bien adapté à ce besoin, grâce à sa conception liée à TSN. Cependant, les observations effectuées grâce à la revue de littérature nous ont permis d'identifier les verrous à lever pour répondre au besoin de synchronisation robuste et précise dans les réseaux TSN embarqués critiques. Ces verrous nous ont amené à réaliser les travaux présentés dans la suite de ce manuscrit. Ils sont présentés dans cette section, en commençant par la précision, puis la robustesse, et enfin l'impact des deux premiers points sur le reste du trafic réseau. Ces trois points sont les trois parties principales du manuscrit.

Pour l'aspect précision, nous profiterons de la disponibilité croissante de matériel supportant TSN pour nous concentrer sur la représentativité du matériel dans les outils de simulation et les méthodes formelles. Ainsi, pour la première partie de ce manuscrit, les objectifs sont i) d'étudier la représentativité d'un outil de simulation à partir de mesures réelles et ii) de proposer une méthode formelle pour dériver une borne peu pessimiste sur la précision dans le pire des cas afin de répondre aux besoins de validation/certification du monde embarqué critique grâce à une modélisation fine des sources d'imprécisions. Ces deux points sont destinés à fournir les clés pour comprendre et explorer les paramètres qui ont un impact sur la précision, aidant ainsi à expliquer les résultats variables obtenus dans les mesures expérimentales.

Pour répondre au besoin de robustesse des réseaux critiques embarqués, la deuxième partie de ce manuscrit commencera par une comparaison entre BMCA et la configuration statique afin de déterminer laquelle de ces deux solutions est la mieux adaptée au réseau critique embarqué que nous étudions. Ensuite, un deuxième chapitre sera consacré à la conception d'une telle configuration statique.

Et enfin, la troisième partie explore l'impact du gPTP sur le trafic utile à travers deux points, mettant en pratique les contributions précédentes. Le premier point est l'évaluation de l'impact de la précision et du trafic gPTP sur le Time-Aware Shaper (TAS), plus précisément la perte de bande passante due au surdimensionnement des fenêtres TAS pour tenir compte de l'imprécision de la synchronisation. Le second point est l'évaluation de l'impact de ces paramètres de dimensionnement de gPTP sur les latences de traversée du réseau dans le pire des cas. Ces deux évaluations seront réalisées dans le cadre d'une application pratique d'un cas d'étude de satellite.

A.2 Précision

A.2.1 Vers une simulation représentative de la réalité

Les premières investigations effectuées pour étudier la précision atteignable avec IEEE802.1AS ont été réalisés par simulation. En effet, la simulation permet d'étudier très finement un protocole : il est possible d'instrumenter n'importe quel type d'événement, d'expérimenter de nombreux paramètres ou de tester de nouveaux mécanismes qui peuvent ne pas être disponibles sur les implémentations matérielles commerciales.

Cependant, bien que la simulation soit un outil très utile, il est nécessaire de s'assurer qu'elle est représentative de la réalité. Dans cette sous-section, nous partons d'une bibliothèque de simulation open source, nous ajoutons les mécanismes gPTP manquants, puis nous l'étalonnons avec du matériel TSN disponible dans le commerce pour nous assurer que les résultats décrivent avec précision une mise en œuvre réaliste. Les résultats confirment la fidélité du simulateur pour les couches physiques 100Base-T et 1000Base-T, comme le montrent les RMSE d'environ 3 ns entre la moyenne glissante de la précision mesurée et simulée.

Pour ce faire, nous sommes partis de la librairie open-source de Puttnies et al. [81] basé sur OMNeT++ et INET. Cependant, avant de la comparer à des implémentations réelles, nous lui avons apporté quelques modifications. La première des modifications est l'implémentation du mécanisme manquant du *rateRatio* afin de respecter le standard. Puis des modifications ont été apportées dans l'objectif d'ajouter des sources d'imprécisions réalistes. La première de ces sources est la granularité de l'horloge. En effet, chaque horloge à un pas qu'il faut prendre en compte pour un horodatage réaliste. La seconde source est la couche physique utilisée. Comme décrit par Loschmidt et al. dans [74], l'implémentation ainsi que le type de couche physique joue un rôle important sur la variabilité du temps de traversée des messages gPTP impactant ainsi la précision du calcul du délai de propagation. À l'aide des travaux de Loschmidt et al., une implémentation représentative des couches physiques 100Base-T et 1000Base-T a été effectué. D'autres sources d'imprécision existe tel que le bruit de l'oscillateur et de la boucle à phase asservie, mais n'ont pas été implémentés à cause leur très faible impact qui rend leur caractérisation très difficile sans matériel de mesure dédiée.

Pour valider que les changements apportés permettent de rendre la librairie représentative de la réalité, une phase d'expérimentation en trois étapes a été réalisé. Les deux premières sont destinées à la calibration des sources d'imprécisions ajoutées et la dernière à la quantification de la représentativité. Ces expériences sont réalisées avec l'installation expérimentale décrite dans la Figure A.6. On y retrouve quatre commutateurs TSN connecté en chaine. Le premier de la chaine joue le rôle de Grandmaster. Les configurations ainsi que la récupération des résultats sont effectués par lien UART depuis un ordinateur Ubuntu. La mesure de la progression des horloges est effectuée à l'aide d'un PPS analyseur. Dans un premier temps, les expériences présentées seront effectuées avec du 100Base-T. Les résultats obtenus avec du 1000Base-T seront discutés plus brièvement à la fin de la sous-section.

La première expérience de calibration a pour objectif de calibrer les horloges simulée, mais surtout de valider la représentativité du modèle d'horloge utilisé par Puttnies et al.. En effet, un simple modèle d'horloge à dérive constante est implémenté dans la librairie. Ainsi, des mesures de dérive d'horloge ont été effectuées en ayant préalablement désactivé la synchronisation pendant une heure. Les résultats obtenus pour deux commutateurs sont présentés dans la Figure A.7. On y observe une dérive linéaire des horloges, qui est confirmée par les régressions linéaires qui permettent aussi d'obtenir le coefficient directeur utile à paramétrer les horloges simulées. La Figure A.8 montre l'évolution au cours du temps de ce coefficient directeur. Cette figure a été obtenue en répétant 24 fois cette expérience d'une heure. On y observe une très faible évolution du coefficient directeur au

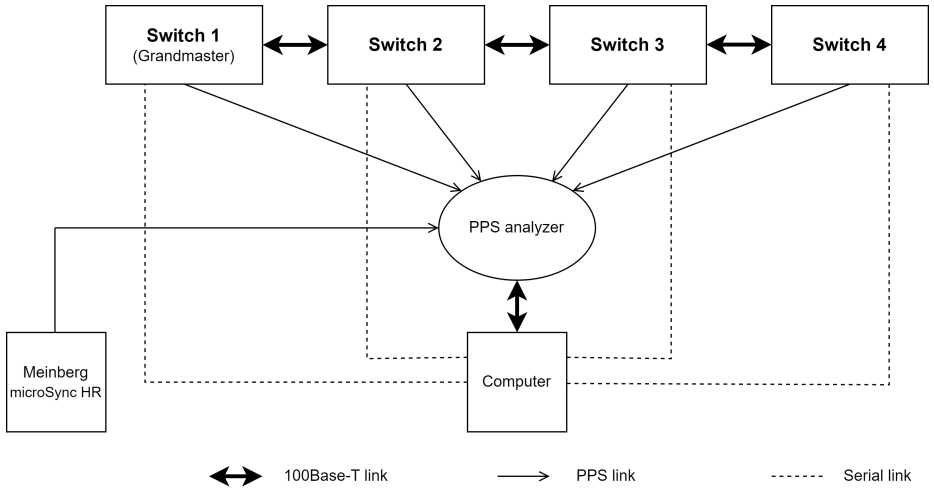


Figure A.6: Topologie expérimentale.

cours de la journée. Cette variation s'explique par les variations de température dans le laboratoire. Cette première expérience nous permet de valider qu'une simulation d'horloge à dérive constante est représentative de nos horloges de notre matériel sur des expériences d'une heure si les variations de température sont faibles.

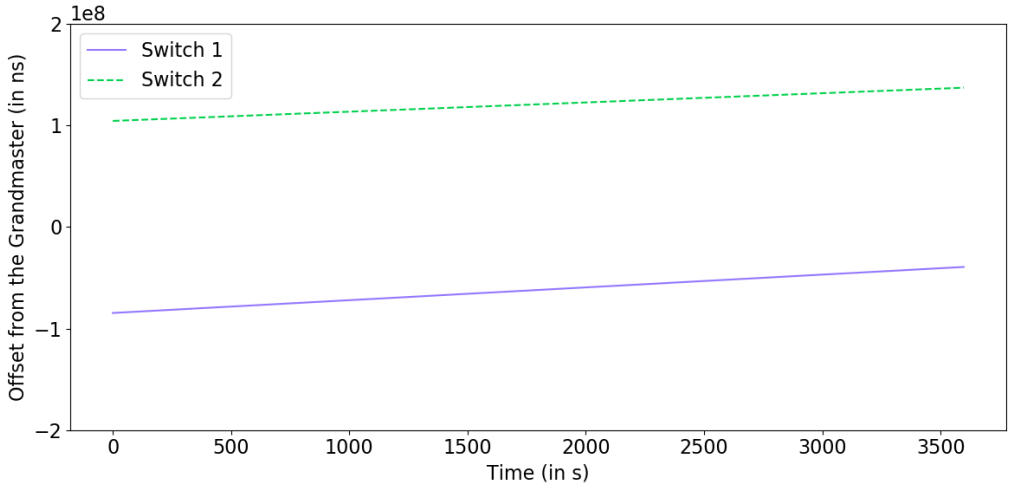


Figure A.7: Dérive de deux horloges de commutateur TSN

La deuxième expérience a pour objectif premier la calibration des sources d'imprécision liée à la couche physique. Pour ce faire, les valeurs de *pdelay* ont été acquises pendant 32h entre deux commutateurs de la chaine. En comparant la distribution obtenue expérimentalement à celle obtenue avec le simulateur pour différentes valeurs des paramètres dimensionnant des imprécisions, nous avons pu minimiser l'erreur quadratique moyenne entre les distributions. Figure A.9 présente la

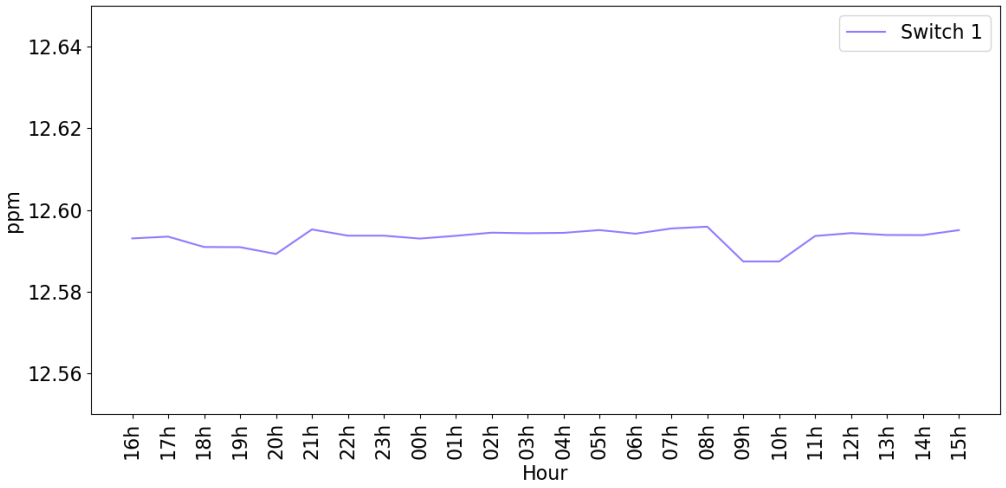


Figure A.8: Résultats de la régression linéaire du commutateur 1 mesuré pendant 24 expériences d'une heure.

distribution des valeurs de *pdelay* mesurées et simulées après calibration. On y observe une très bonne représentativité des valeurs de *pdelay* obtenu par le simulateur. On note toutes fois un léger pessimisme du simulateur quant aux valeurs maximales, mais qui n'est pas un problème dans un contexte critique.

La dernière expérience a pour objectif de valider la représentative du simulateur en termes de précision après calibration. Pour ce faire, la précision a été mesurée expérimentalement 19 fois par série d'une heure. Figures A.10, A.11 et A.12 comparent la précision simulée et la précision mesurée. On observe que les extrêmes obtenus en simulation bornent très finement la précision expérimentale. Cette observation est confirmée par la racine de l'erreur quadratique moyenne comprise entre 2.9 ns et 4.2 ns.

Ce plan d'expérimental a aussi été répété avec le même réseau, mais avec des liens 1000Base-T afin de montrer son adaptabilité à différentes couches physiques. La première série de mesures pour calibrer l'horloge n'a pas été nécessaire, car les commutateurs utilisés sont identiques. La deuxième série de mesures pour calibrer les sources d'imprécision du canal de communication a mis en évidence une différence de l'ordre de quelques dizaines de nanosecondes de variation du délai entre horodatage d'envoi et de réception entre les deux couches physique. Figure A.13 compare la distribution des *pdelay* expérimentale avec la distribution simulée après calibration. La dernière série de mesures a permis de valider la représentativité en termes de précision comme le montre la Figure A.14. Cette représentativité est confirmée par une racine de l'erreur quadratique moyenne de 3.2 ns entre la simulation et les mesures.

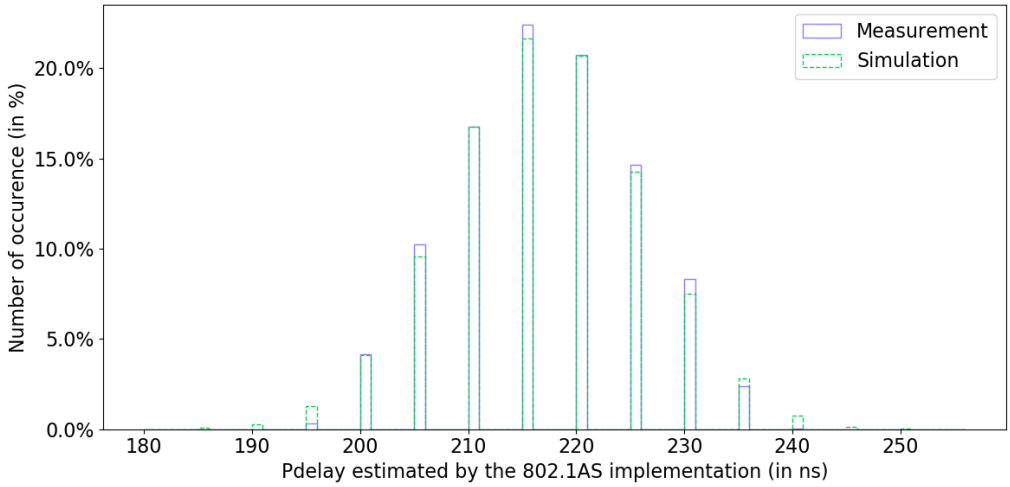


Figure A.9: Distribution des résultats obtenus par les mécanismes *Pdelay* simulé et réel pendant 32 heures

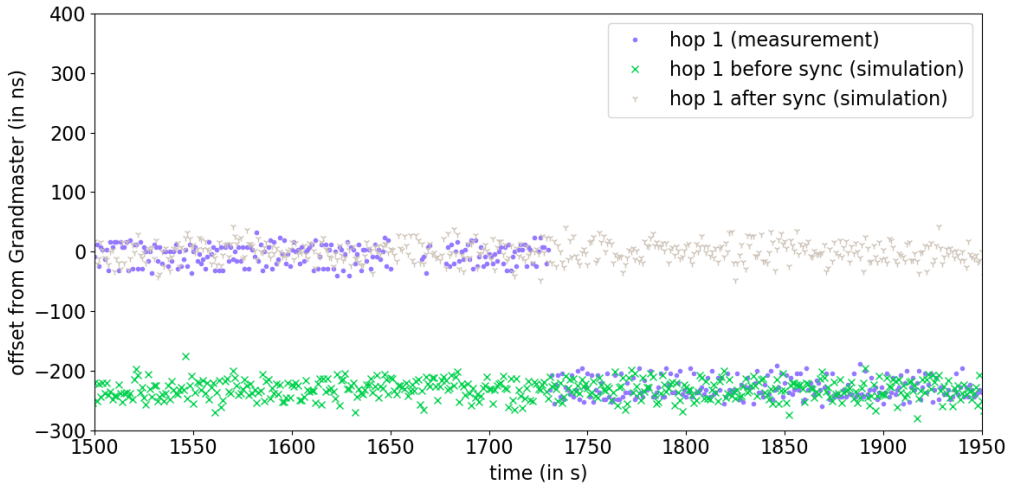


Figure A.10: Mesure de précision entre le commutateur du saut 1 et le Grandmaster. Les pires et meilleures précisions sont aussi affichés

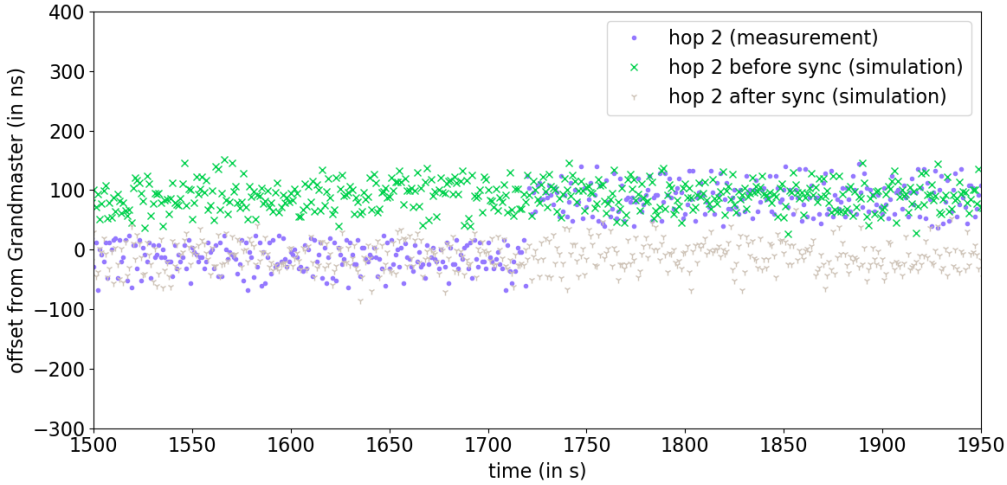


Figure A.11: Mesure de précision entre le commutateur du saut 2 et le Grandmaster. Les pires et meilleures précisions sont aussi affichés

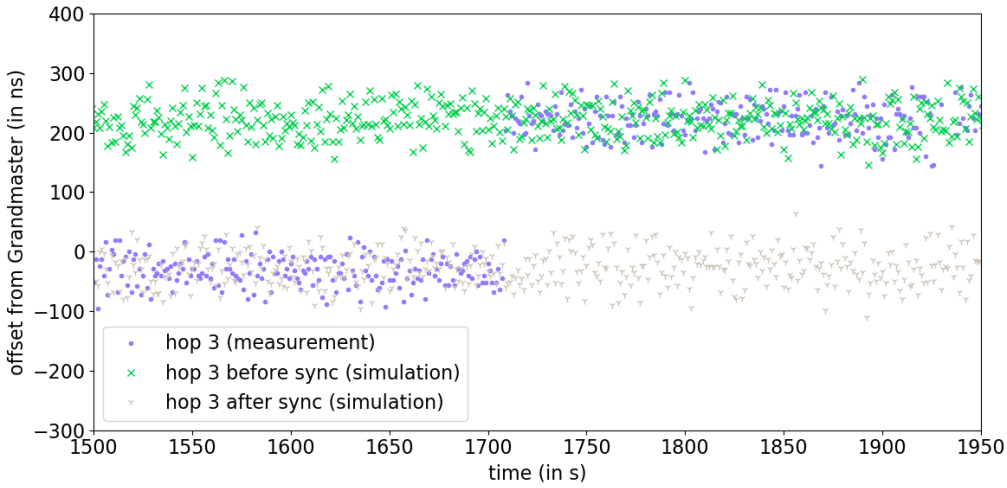


Figure A.12: Mesure de précision entre le commutateur du saut 3 et le Grandmaster. Les pires et meilleures précisions sont aussi affichés

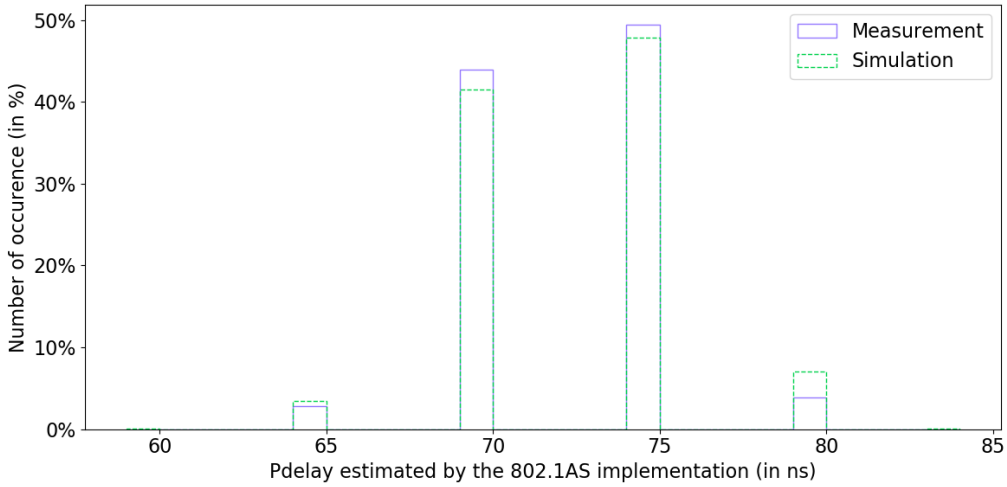


Figure A.13: Distribution des *pdelay* mesuré et simulé après calibration avec la couche physique 1000Base-T

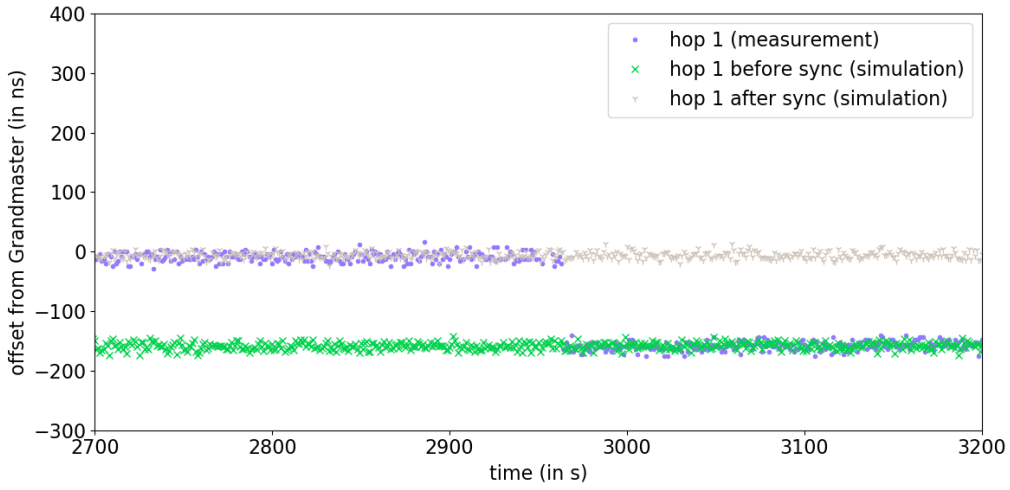


Figure A.14: Mesure de précision entre le commutateur du saut 1 et le Grandmaster avec la couche physique 1000Base-T. Les pires et meilleures précisions sont aussi affichés

A.2.2 Une borne sur la précision de synchronisation pire cas

Un simulateur représentatif est un outil puissant pour étudier gPTP mais n'est pas suffisant pour la validation/certification d'un réseau critique embarqué. En effet, pour un tel réseau, il est nécessaire d'étudier ce qu'il se passe dans le pire cas. Ainsi, pour un protocole de synchronisation, une étude de la précision pire est incontournable. [62] propose un modèle formel pour calculer une borne sur ce pire cas pour le 100Base-T. Nous proposons dans cette sous-section d'améliorer ce modèle en le rendant moins pessimiste, en le généralisant pour supporter n'importe quelle couche physique et en l'étendant pour aussi obtenir une borne inférieure. Pour étayer le bon fonctionnement de ce modèle, ces résultats seront comparés à des mesures expérimentales, des simulations représentatives ainsi que de la recherche exhaustive de la pire précision possible. De plus, ces travaux fournissent les clés pour quantifier l'impact des nombreux paramètres impactant la précision de synchronisation.

Commençons par décrire brièvement les développements effectués pour arriver à ce nouveau modèle. Ce dernier repose sur une décomposition du protocole en suite d'équation qui doivent être calculées afin d'obtenir la correction a appliqué à l'horloge. Pour chacune de ces équations, chaque paramètre est étudié afin de comprendre quelles sources d'imprécision peuvent l'impacter. Ensuite, ces différentes sources d'imprécision sont utilisées pour maximiser ou minimiser le résultat de chaque équation afin de conduire à la pire précision possible. Prenons l'exemple du calcul du *neighborRateRatio* dont l'équation est (A.2). Cette équation dépend des horodatages t_3 et t'_3 qui sont pris par l'équipement répondeur et servent à calculer la durée $t'_3 - t_3$. Ainsi cette durée ne peut que subir des imprécisions liées à la granularité de l'horloge comprise en -10 ns et +10 ns avec notre matériel. De l'autre côté du lien, en plus d'être impacté par la granularité comme $t'_3 - t_3$, $t'_4 - t_4$ est aussi impacté par la variabilité du temps de traversée du lien causé par la couche physique et son implémentation. En effet, un premier message peut arriver plus rapidement que le deuxième, venant ainsi "compresser" la durée. En prenant en compte toutes ces sources d'imprécision, il est possible d'en déduire l'erreur minimale ou maximale possible pour ce mécanisme. En répétant ce processus pour chacune des équations du protocole, on peut ainsi maximiser ou minimiser le résultat de l'équation (A.5). Cette sur ou sous-estimation du résultat de cette équation a pour conséquence une sous ou sur-correction de l'horloge comme illustré dans la Figure A.15. Ainsi, en déterminant ses bornes minimale et maximale et en l'ajoutant à la dérive minimale ou maximale de l'horloge, on obtient une borne sur la précision minimale et maximale.

Le développement conduit alors aux équations résumées dans les tableaux A.1 pour la borne supérieure et A.2 pour la borne inférieure avec i, l'indice de l'équipement, ρ la borne sur la dérive de l'horloge, I_s la période entre l'envoi de deux messages `Sync`, J_{fup} la borne sur la gigue réseau des messages `Follow_Up`, G la granularité de l'horloge, $d^{min}_{(i)\to(j)}$ le délai de propagation de l'horloge de l'équipement i à l'horloge de l'équipement j, τ la borne sur le temps de résidence entre la réception d'un message et l'envoi de la réponse, A la borne sur l'asymétrie induite par la couche physique, $J_{i\to j}$ la borne sur la gigue induite par la couche physique et I_p la période entre l'envoi de deux messages `Pdelay_Req`.

Maintenant que nous avons un modèle, commençons par le comparer au modèle de l'état de l'art, à notre simulateur et à la recherche exhaustive afin d'évaluer son pessimisme. Pour ce faire, nous utiliserons le paramétrage par défaut du protocole, les valeurs de source obtenu lors de la calibration du simulateur pour le 100Base-T et nous supposerons que les horloges ont une dérive bornée entre -10 ppm et 10 ppm. La Figure A.16 montre les résultats de cette comparaison sur un réseau de taille représentative d'un réseau embarqué, soit jusqu'à 10 sauts, et sur un réseau de très grande taille, soit 100 sauts. On y observe que notre modèle conduit à un pessimisme plus faible que le modèle de l'état lors de la comparaison avec le simulateur. On observe aussi que plus le réseau est grand, plus l'écart se creuse. Sur les deux premiers sauts, on note que notre modèle suit finement les résultats

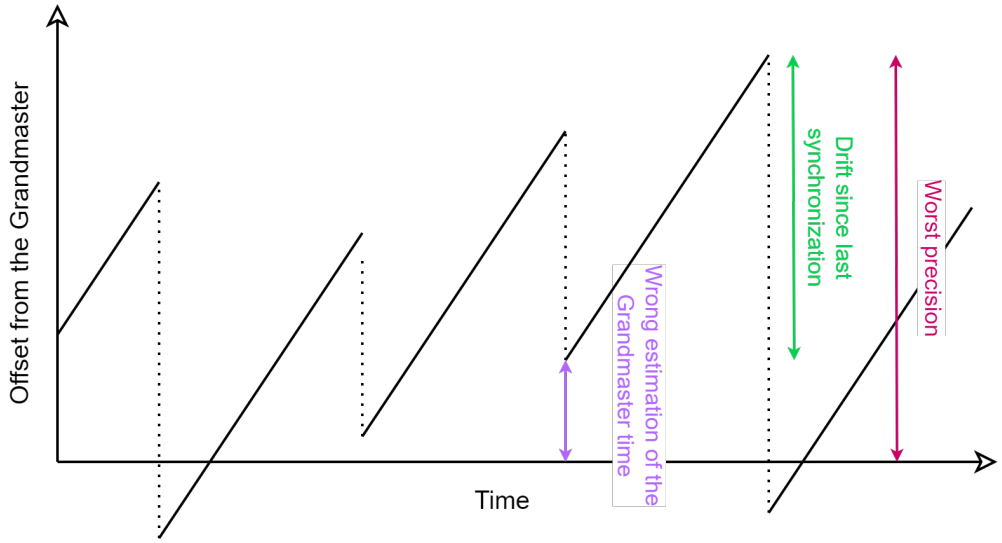


Figure A.15: Illustration des deux quantités qui impacte la précision entre le Grandmaster et un équipement gPTP en fonction du temps

P_i^U	$(\lvert\rho_i\rvert + \lvert\rho_{GM}\rvert)(I_s + J_{fup}) + \delta GM_i^U$
δGM_i^U	$\delta C_{i-1}^U + \delta D_i^U + G$
δC_{i-1}^U	$\delta D_{i-1}^U(\frac{nr^{i-2}-1}{nr-1}) + G(\frac{nr^{i-1}-1}{nr-1} - 1) + \delta nr^U((d^{\min}_{(i-2)\to(i-1)} + \delta D_{i-1}^U)\sum_{j=0}^{i-2} j \times nr^{j-1} +$ $(\tau_{i-1} + G)\sum_{j=1}^{i-1} j \times nr^{j-1})$
δD_i^U	$\frac{[(\tau_i + 2d_{i\to j}^{min} + J_{j\to i} + J_{i\to j} + A)(\rho_i+1)+G](nr+\delta nr^U) - (\tau_i(1-\rho_j)-G)}{2} - (d^{\min}_{j\to i})$
δnr^U	$\frac{[2G + J_{j\to i}(1+\rho_j) + G(\rho_j - \rho_i)]}{I_p(1-2\rho_i+\rho_i^2)+(\rho_i-1)(G+J_{j\to i})}$

Table A.1: Formules de calcul de la borne supérieur sur la précision de synchronisation pire cas

obtenus par la recherche exhaustive. Ces résultats sont détaillés sur la Figure A.17. Cette figure met en avant l'optimisme de la solution de l'état de l'art. En effet, on observe qu'une précision supérieure a été mesurée dans le simulateur. Un faible pessimiste de 5.4% est mesuré entre la borne obtenue par notre modèle et le résultat obtenu par recherche exhaustive du pire cas. Pour la borne inférieure, ce pessimisme n'est que de 9.9%.

Figure A.18 montre la comparaison entre les bornes et les précisions minimale et maximale mesurée expérimentalement sur un réseau à quatre commutateurs. Comme pour les figures précédentes, on y observe un pessimisme réduit qui augmente avec le nombre de sauts. Ceci est dû à la suite d'événements nécessaires pour observer une valeur proche de la précision minimale ou maximale qui devient de moins en moins probable de par sa complexité.

Des résultats similaires ont été obtenus avec le 1000Base-T avec un pessimisme allant jusqu'à 20.5% pour la borne haute et 19.8% pour la borne basse par rapport à la recherche exhaustive. La comparaison illustrée dans la Figure A.3 met en avant que, malgré le pessimisme supérieur, une configuration avec du 1000Base-T est plus précise qu'une configuration 100Base-T dans le pire cas.

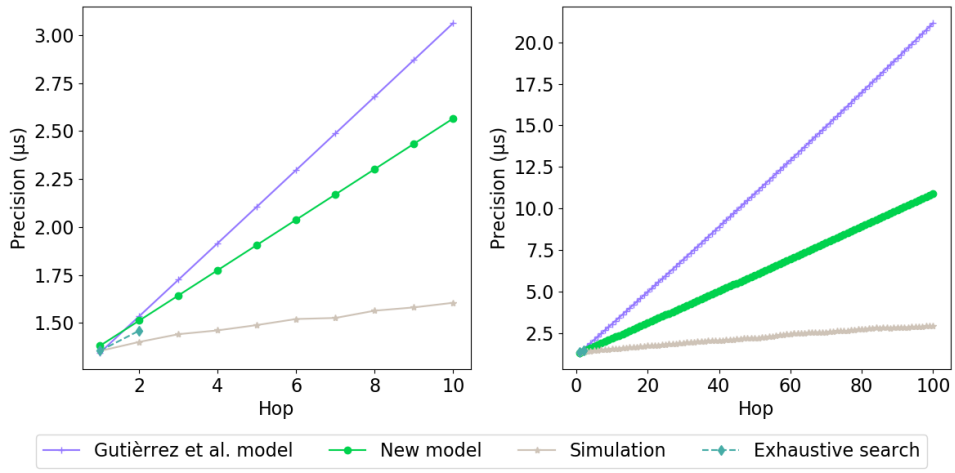


Figure A.16: Comparaison entre la simulation, la recherche exhaustive et la borne avec la couche physique 100Base-T en fonction du nombre de sauts.

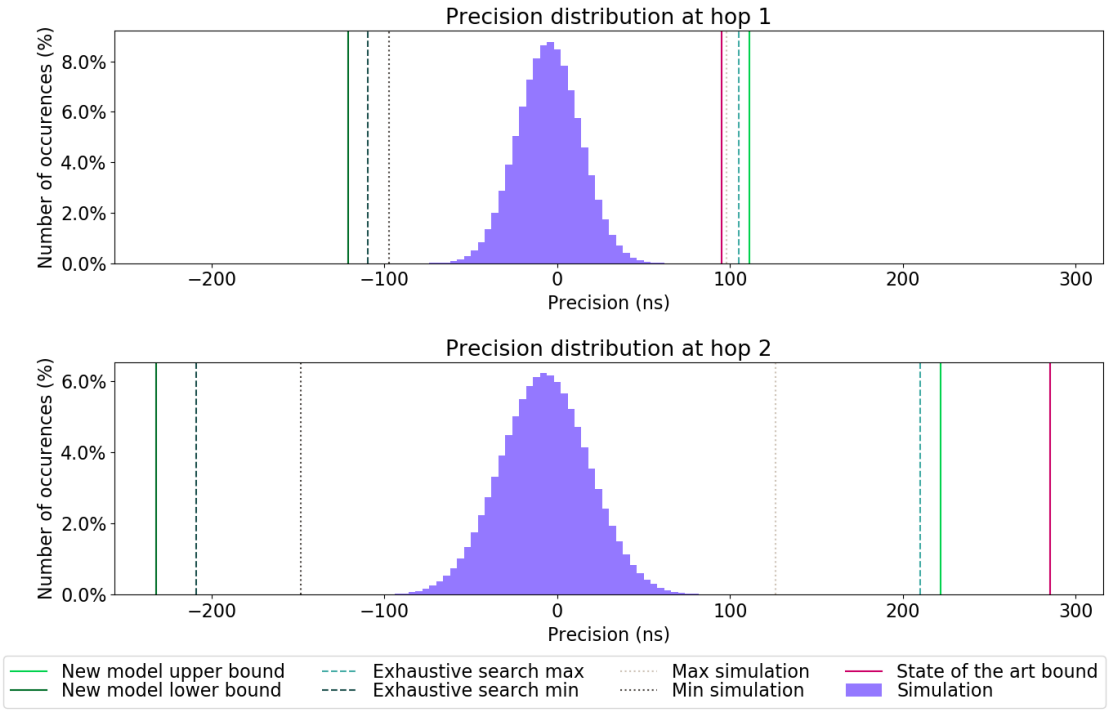


Figure A.17: Comparaison entre la simulation, la recherche exhaustive et la borne avec la couche physique 100Base-T au saut 1 et 2.

P_i^L	$-(\lvert\rho_i\rvert + \lvert\rho_{GM}\rvert)(I_s + J_{fup}) + \delta GM_i^L$
δGM_i^L	$\delta C_{i-1}^L + \delta D_i^L - 2G$
δC_{i-1}^L	$\delta D_{i-1}^L(\frac{nr^{i-2}-1}{nr-1}) - G(\frac{nr^{i-1}-1}{nr-1} - 1) + \delta nr^L((d_{(i-2)\to(i-1)}^{\max} + \delta D_{i-1}^L)\sum_{j=0}^{i-2} j \times nr^{j-1} + (\tau_{i-1} - G)\sum_{j=1}^{i-1} j \times nr^{j-1})$
δD_i^L	$\frac{[(\tau_i+2d_{i\to j}^{min}+A)(1-\rho_i)-G](nr+\delta nr^L)-(\tau_i(1+\rho_j)+G)}{2} - (d_{i\to j}^{\min} + J_{j\to i} + A)$
δnr^L	$\frac{-[2G+J_{j\to i}(1-\rho_j)+G(\rho_i-\rho_j)]}{I_p(1+2\rho_i+\rho_i^2)+(\rho_i+1)(G+J_{j\to i})}$

Table A.2: Formules de calcul de la borne inférieur sur la précision de synchronisation pire cas

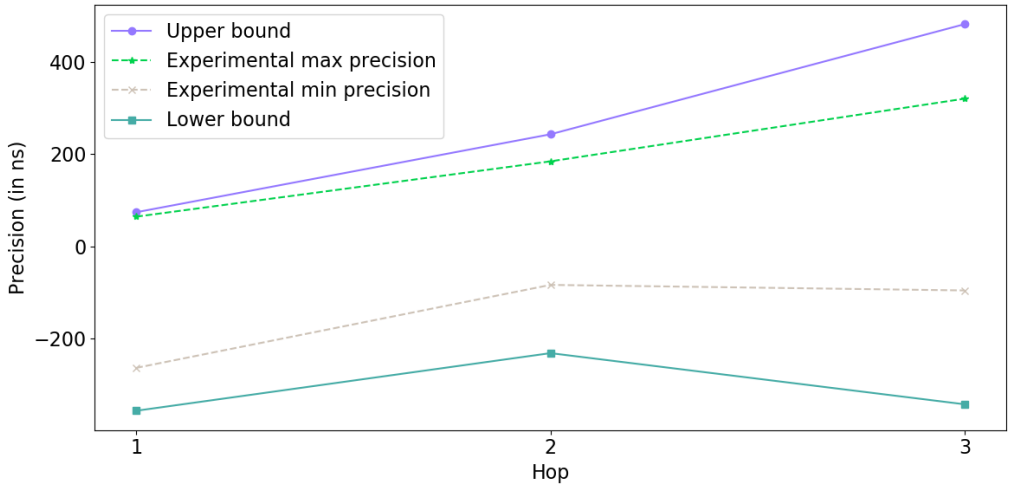


Figure A.18: Comparaison entre la précision expérimentale et les bornes haute et basse avec la couche physique 100Base-T en fonction du nombre de saut.

Cela s'explique par la gigue et l'asymétrie induites par la couche physique qui sont plus faibles dans le cas du 1000Base-T qu'avec le 100Base-T sur le matériel que nous utilisons.

Suite à cette étape de validation, le modèle à était utilisé afin d'étudier l'influence des différents paramètres sur la précision. Figure A.20 illustre cette analyse sur les termes de l'équation finale (5.23). On observe que la majeure partie la pire précision est causé par la dérive des horloges du Grandmaster et de l'équipement au saut i. De plus, même en cas d'utilisation d'horloge hypothétiquement parfaite que ne dériverait pas, on observe que la pire précision possible augmente en fonction du nombre de sauts à cause de l'accumulation d'erreur à chaque saut. Dans ce cas, la borne passe de 2.17μs à 0.92μs au septième saut. Ces 0.92μs peuvent être réduits en jouant par exemple sur les paramètres restant comme ceux liés à la couche physique ou bien la granularité des horloges. Figure A.21 plonge plus dans les détails en étudiant indépendamment l'influence de chaque paramètre. On y observe que l'on peut regrouper les paramètres en trois catégories par l'influence. Le *syncInterval* I_s, la qualité des horloges Grandmaster ρ_{GM} et de l'équipement i ρ_i et le nombre de sauts i ont un impact considérable. De leur côté, la gigue réseau des `Follow_Up` J_{fup}, la granularité G et les paramètres de la couche physique J et A ont une influence plus faible

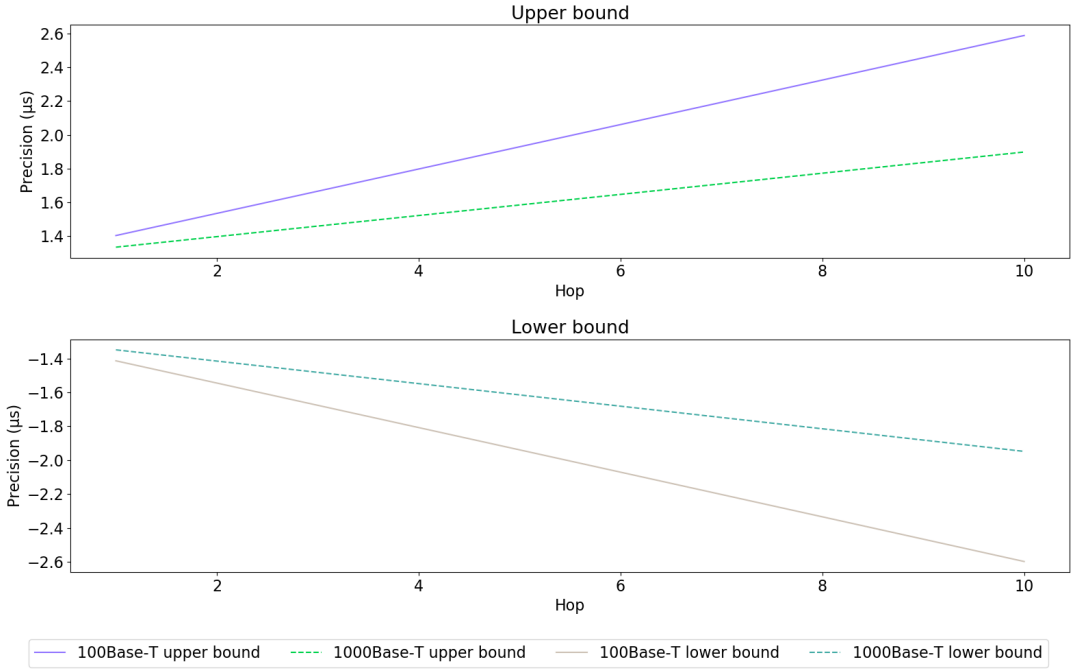


Figure A.19: Comparaison des bornes 100Base-T et 1000Base-T

sur la borne, mais reste des leviers intéressants lorsque les paramètres précédents ont atteint leur limite. Les paramètres restants ont un impact négligeable sur la borne. Il y a donc peu d'intérêt à les estimer finement pour calculer les bornes sur la précision.

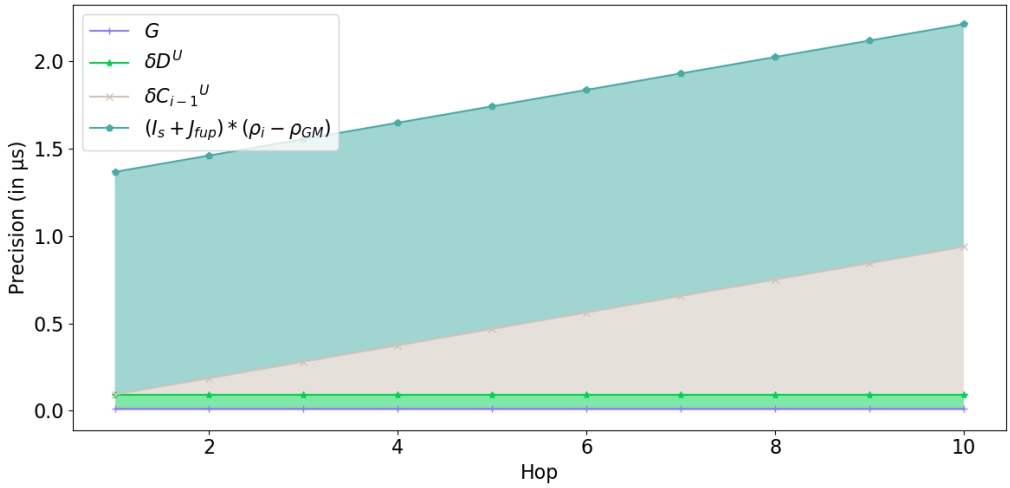


Figure A.20: Influence de chaque terme de Eq (5.23) sur la borne supérieur de precison sur un réseau 100Base-T en fonction du nombre de saut.

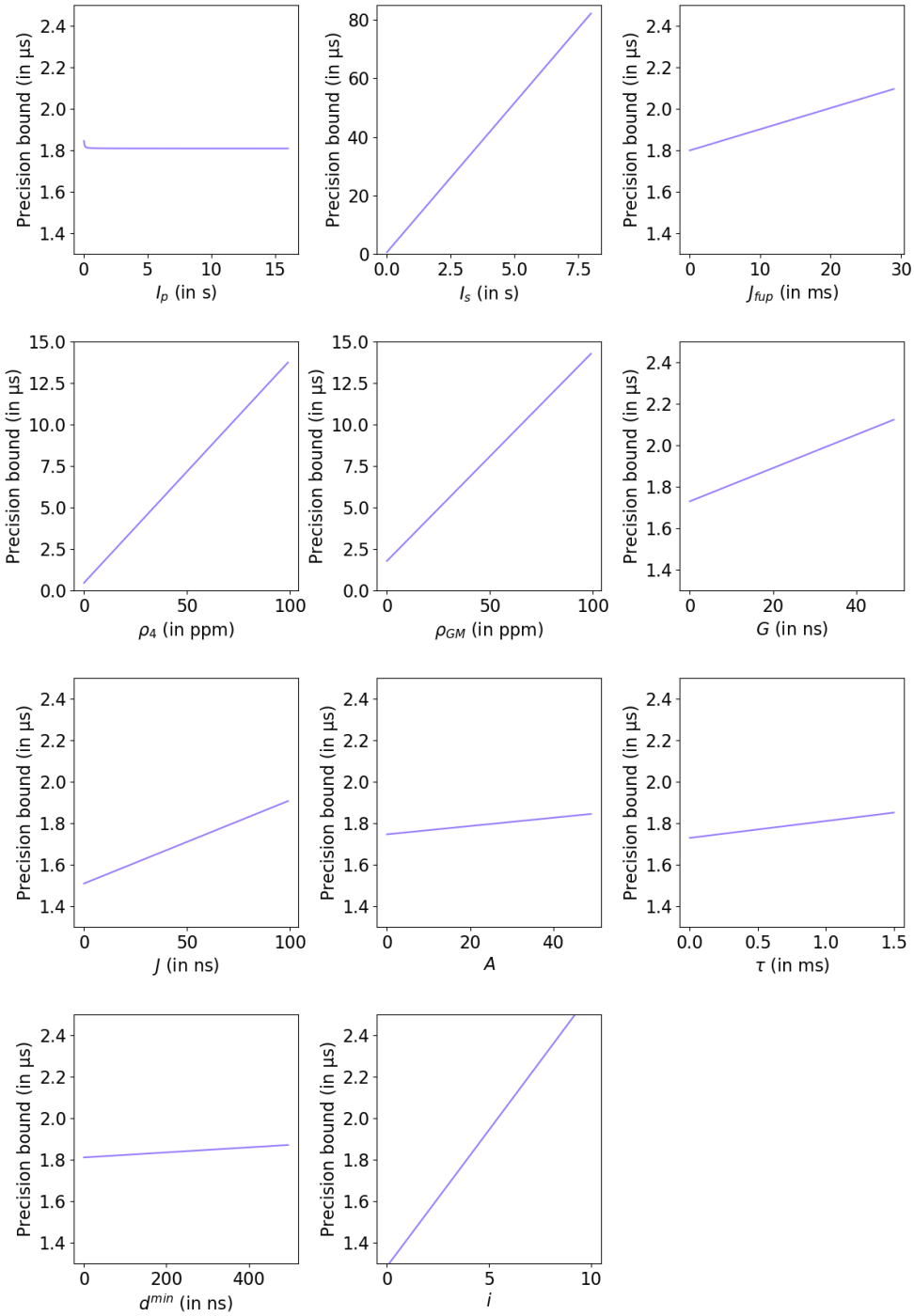


Figure A.21: Influence des paramètres du calcul de la borne supérieur sur la précision avec du 100Base-T.

A.3 Robustesse aux pannes

A.3.1 BMCA versus configuration statique à multiples domaines

IEEE802.1AS propose deux mécanismes pour rendre la synchronisation robuste aux pannes : le BMCA et la configuration statique à multiple domaine. Dans cette section, nous comparons ces deux mécanismes afin de déterminer le plus adapté aux contraintes de l'embarqué.

Cette comparaison est basée sur cinq métriques qualitatives et quantitative qui ont été choisies pour répondre au besoin des réseaux embarqué critique. Les métriques sont présentées dans la liste suivante :

- Pouvoir de détection et d'atténuation des pannes
- Consommation de bande passante
- Temps de reconfiguration en cas de panne
- Précision pire cas en cas de panne
- Complexité durant la conception

Les résultats pour chaque métrique sont très brièvement détaillés puis sont suivis d'une conclusion.

Dans nos expériences, le pouvoir de détection de pannes est identique. En effet, cette partie du mécanisme n'est pas encore standardisée pour la configuration statique. Nous avons donc choisi d'implémenter le même mécanisme que pour le BMCA soit après trois `Sync` manquant, une panne est détectée. Cette détection déclenche alors l'exécution du BMCA ou la sélection d'un nouveau domaine. Pour ce qui est du pouvoir d'atténuation, le BMCA est généralement bien plus puissant que le mécanisme statique. En effet, tant qu'il existe un chemin, le BMCA trouvera un moyen d'attendre les nœuds grâce à la découverte du réseau effectué par les messages `Announce` alors que la configuration statique est limitée par le nombre de domaines disponible. Notre étude a montrée que, pour répondre au besoin des cas d'étude utiliser dans ce manuscrit, seul 2 ou 3 domaines par Grandmaster suffit pour répondre au besoin de disponibilité en cas de panne de lien et de commutateur.

Les travaux autour de la bande passante consommée par le protocole de synchronisation, ont mis une très faible consommation pour les deux mécanismes. En effet, celle-ci atteint 0.00130% sur un lien lorsque le BMCA est utilisé et 0.00417% lorsque la configuration statique est utilisée avec quatre domaines.

Le temps de reconfiguration après une panne a été étudié sur deux topologies présentées dans la Figure A.22. Sur ces deux topologies, le temps de reconfiguration le plus faible a été mesuré avec la configuration statique. En effet, il était compris entre 252 ms et 399 ms contre 282 ms et 541 ms pour le BMCA. Le temps de reconfiguration de la configuration statique et sa distribution s'explique par le temps de détection de la panne compris entre 2 et 3 périodes d'envoi de `Sync`, soit entre 250 ms et 375 ms, auquel viennent s'ajouter des imprécisions liées à notre instrumentation. Les résultats du BMCA sont plus difficiles à expliquer et soulèvent la question du déterminisme de la solution et surtout de sa preuve. Une autre série de mesures a aussi montré qu'en cas de panne de Grandmaster avec la configuration statique et un Grandmaster de secours en hot-standby, le temps de reconfiguration est, lui aussi, plus faible qu'avec le BMCA, entre 253 ms et 412 ms, et facilement explicable.

Avec un temps de reconfiguration plus long, les horloges ont plus de temps pour dériver. Ceci induit inévitablement une moins précision en cas de panne pour le BMCA. Dans ce paragraphe, nous nous intéressons à ce qu'il se passe dans le pire cas. Les résultats suivants ont été obtenus en prenant en compte le temps de reconfiguration mesuré dans le calcul des bornes pires cas détaillé dans la section précédente. Pour le BMCA, la borne supérieure est de 5.6 µs contre 4.3 µs pour

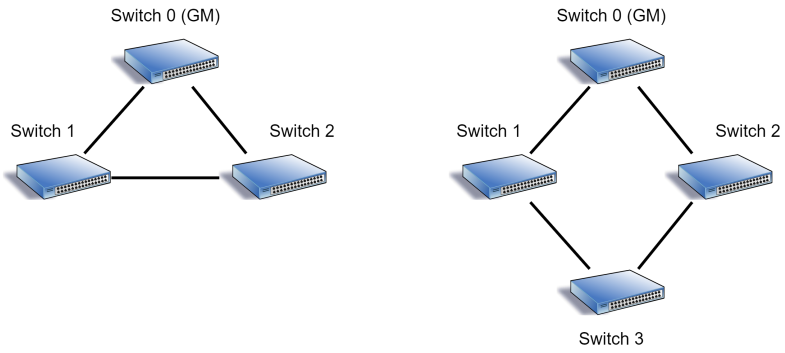


Figure A.22: Topologies utilisées pour étudier le temps de reconfiguration des deux mécanismes de robustesse de IEEE802.1AS

la configuration statique. Cette différence peut être facilement compensée au prix d'horloge de meilleures qualités ou bien d'un *syncInterval* plus faible.

Et enfin, nos recherches sur la phase de conception ont montré que le BMCA est bien plus simple à configurer. En effet, ce mécanisme ne laisse au concepteur du réseau le contrôle que sur l'ordre des Grandmasters à utiliser à l'aide d'un paramètre appelé *priority1*. Ceci en fait une solution "plug and play". Alors que pour le mécanisme statique, le concepteur doit déterminer l'état de chaque port gPTP afin de créer l'arbre couvrant de distribution de chaque domaine. Et ceci en se demandant quels arbres sont les plus adaptés pour mitiger chaque possible panne, ou bien si un arbre est plus précis qu'un autre, ou encore ou placer le Grandmaster dans le réseau pour optimiser la robustesse et/ou la précision.

Cette étude nous permet de conclure que, bien qu'offrant des performances très similaires sur les topologies testées, le BMCA n'est pas adapté au monde des réseaux embarqués à cause de son déterminisme qui reste à prouver. En effet, la configuration statique ne consomme que très peu de bande passante supplémentaire et a un pouvoir d'atténuation qui peut égaler le BMCA sur réseau relativement petit du monde de l'embarqué. Son défaut majeur reste la conception de la configuration optimale qui peut être complexe, mais il est courant dans les systèmes critiques d'avoir une phase de conception complexe pour un fonctionnement simple et déterministe.

A.3.2 Conception d'une configuration statique à multiples domaines robuste et précise

Dans cette section, nous proposons une méthode de conception de configuration statique qui adresse le défaut majeur de ce mécanisme : la complexité de trouver une configuration qui répond au besoin. À la fin de cette section, l'utilisation des métriques, élaboré pour la méthode, seront détournés pour proposer des optimisations de la topologie ou bien du placement des Grandmaster afin d'améliorer la précision de synchronisation et/ou la robustesse aux pannes.

La méthode proposée est décrite dans la Figure A.23. Elle prend pour paramètres d'entrée la topologie, ainsi que les Grandmasters et le nombre de domaines désiré pour chacun. Elle retourne la configuration d'arbre couvrant la plus précise parmi les plus robustes. Chaque fonction est détaillée dans les paragraphes suivants.

La première fonction consiste à générer l'ensemble des arbres couvrant pour un Grandmaster. Elle implémente une méthode commune qui repose sur la génération de toutes les combinaisons

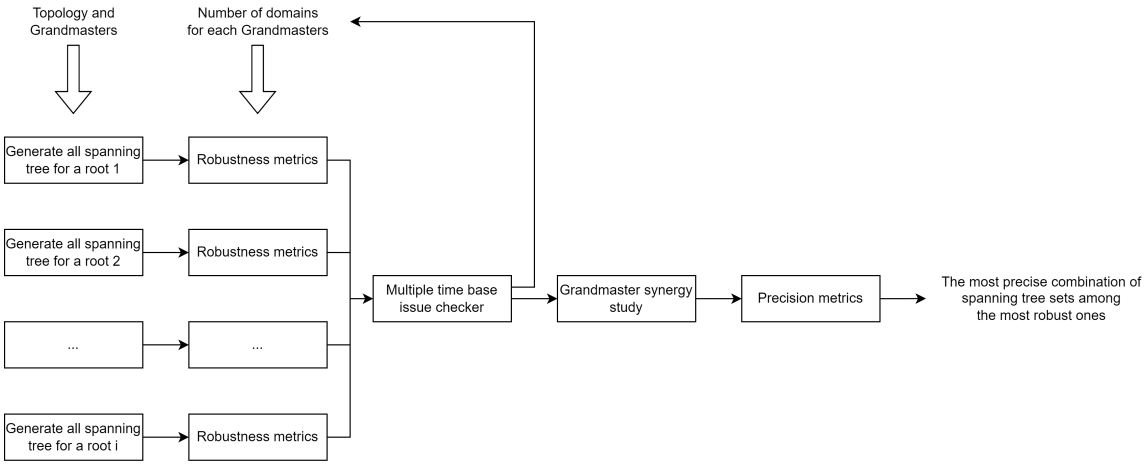


Figure A.23: Illustration de la méthode proposée pour concevoir une configuration statique robuste et précise.

possibles de $n-1$ lien, n étant le nombre d'équipements dans le réseau, avant de supprimer les combinaisons qui ne sont pas un arbre couvrant.

La deuxième fonction a pour objectif de sélectionner les ensembles d'arbres couvrants d'un Grandmaster les plus robustes aux pannes. Pour ce faire toutes les combinaisons possibles de x arbres couvrants, x étant le nombre de domaines pour le Grandmaster en question, sont évaluées par une métrique de robustesse. À l'issue de l'évaluation, chaque ensemble obtient un score. Les ensembles ayant le meilleur score sont sélectionnés pour passer à l'étape suivante. Cette fonction ainsi que la précédente sont exécutées pour chaque Grandmaster.

La troisième fonction permet de vérifier si la configuration est propice à l'apparition d'un cas de panne appelé problème des multiples base de temps. Si c'est le cas, la fonction recommande à l'utilisateur d'augmenter le nombre de domaines pour certain Grandmaster. Cette étape reste optionnelle, car conduit à un sur-dimensionnement de la configuration pour résister à plus de panne que spécifié.

La quatrième fonction est une nouvelle étape d'évaluation de la robustesse, mais cette fois porte sur les combinaisons des ensembles d'arbres couvrants. Ainsi, chaque combinaison est évaluée à l'aide de la métrique de robustesse. Les combinaisons avec le meilleur score sont sélectionnées pour l'étape suivante.

La dernière fonction évalue la précision des différentes combinaisons restantes. Pour ce faire, cette fonction repose sur une métrique d'évaluation de la précision qui est liée au nombre de sauts nécessaire pour atteindre chaque appareil depuis le Grandmaster. En sortie de cette fonction, on obtient les combinaisons d'ensemble d'arbre couvrant les plus précises parmi les plus robustes.

L'application de cette méthode sur un réseau embarqué automobile à permis de réduire l'ensemble des configurations possibles à 2 Grandmaster avec 2 domaines chacun, de 4356 à 8 en 2min31. Dans le cas d'une application similaire sur le réseau AFDX de l'A350, l'unique configuration optimale est déterminée en 8h parmi les 157 778 721 configurations possibles.

En plus de cette méthode, nous proposons d'utiliser les métriques afin d'optimiser le placement des Grandmasters pour maximiser la robustesse et/ou la précision. Pour ce faire, nous itérons sur l'ensemble des placements possible et calculons le score à la métrique. Le meilleur résultat à la métrique considérée donne le meilleur placement pour répondre au besoin. Ces méthodes

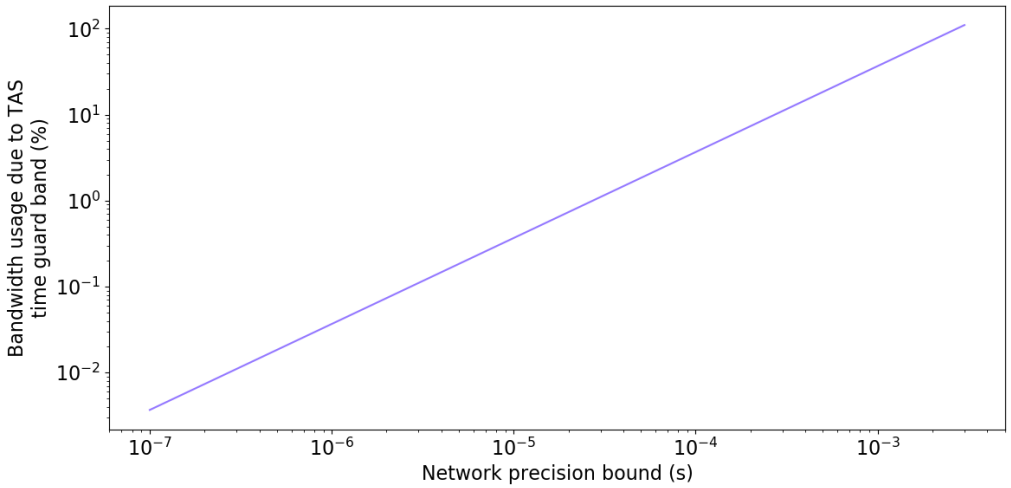


Figure A.24: Bande passante utilisée par les bandes de garde temporelle en fonction de la précision de synchronisation avec 184 fenêtre TAS par seconde.

d'optimisations ont aussi été étendues à la topologie en permettant de trouver l'endroit optimal où ajouter un lien pour améliorer la robustesse et/ou la précision.

A.4 Synchronisation et les autres activités réseau

Dans les sections précédentes, nous avons vu que la précision est hautement configurable à l'aide de paramètre comme la période de synchronisation et que la robustesse l'est aussi à l'aide de paramètre comme le nombre de domaines. Dans cette section, nous étudions l'impact de ces paramètres sur le reste du réseau en appliquant nos résultats sur un cas d'étude de réseau TSN embarqué dans un satellite.

A.4.1 Impact du protocole de synchronisation sur la bande passante

Nous avons vu précédemment que la consommation de bande passante du protocole est très faible même avec plusieurs domaines qui cohabitent sur un même lien. Cependant, la synchronisation a aussi un impact sur la bande passante à travers le surdimensionnement qui doit être effectué pour chaque fenêtre TAS afin de prendre en compte les imprécisions de synchronisation. Ce surdimensionnement rend cette partie de la bande passante inutilisable. Dans cette sous-section, nous quantifions l'impact de ce sur-dimensionnement sur la bande passante afin de montrer qu'il est relativement petit.

Sur le cas d'étude du satellite, le lien le plus demandeur de fenêtre TAS en possède 184 par seconde. Chacune de ces fenêtres temporelles doit être allongée au début et à la fin de la durée de la précision. Ainsi, en fonction de la précision, la part de la bande passante utilisable évolue comme le montre la Figure A.24. On observe sur cette figure qu'une précision de 3 ms conduirait à utiliser toute la bande passante disponible pour les bandes de garde, rendant le réseau inutilisable.

Nous savons des sections précédentes que des précisions de l'ordre de quelques microsecondes sont atteignables avec IEEE802.1AS. Ainsi, nous avons concentré notre étude autours de ces valeurs

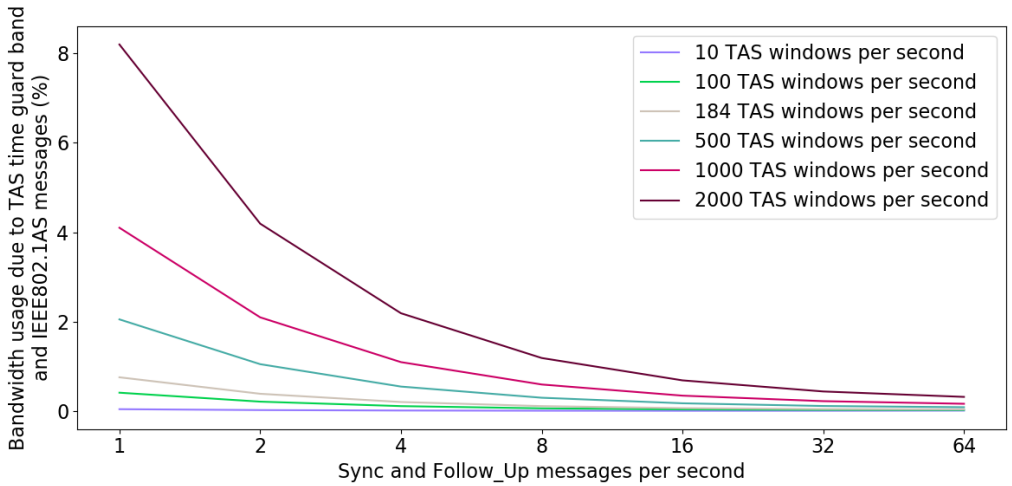


Figure A.25: Bande passante utilisée par les bandes de garde temporelle en fonction de la fréquence de synchronisation pour différent nombre de fenêtres TAS par seconde.

de précision. Figure A.25 illustre cette consommation de bande passante en fonction de la période d'envoi des `Sync`. On observe que plus le nombre de fenêtres TAS par seconde est important, plus la bande passante consommée est grande. Cependant, même avec 2000 fenêtres TAS par seconde, en synchronisant huit fois par seconde, la bande passante utilisé reste en dessous des 2%. Sur les cas moins prospectif, soit les 500 fenêtres par seconde, la consommation est bien inférieure à 1% avec les paramètres par défaut de gPTP.

La multiplication de domaine augmente la bande passante consommée comme le montre la Figure A.26. Cependant, cet effet n'est observable que lorsque la précision est faible (à gauche du graphique) soit quand le surdimensionnement des fenêtres est très faible. Lorsque la précision est moins bonne (à droite du graphique), la consommation de bande passante est causée par le surdimensionnement important. On voit ainsi apparaitre un optimal qui montre qu'améliorer la précision en multipliant les messages de synchronisation peut entrainer une plus grande consommation que ce que le gain de précision apporte au surdimensionnement des fenêtres TAS. Cet optimal est d'autant plus rapide à atteindre lorsque le nombre de domaines est important à cause du trafic supplémentaire.

Malgré ces observations, il est important de souligner que la consommation de bande passante causée par les messages gPTP et le surdimensionnement des fenêtres TAS est très faible même avec plusieurs domaines et un intervalle de synchronisation faible.

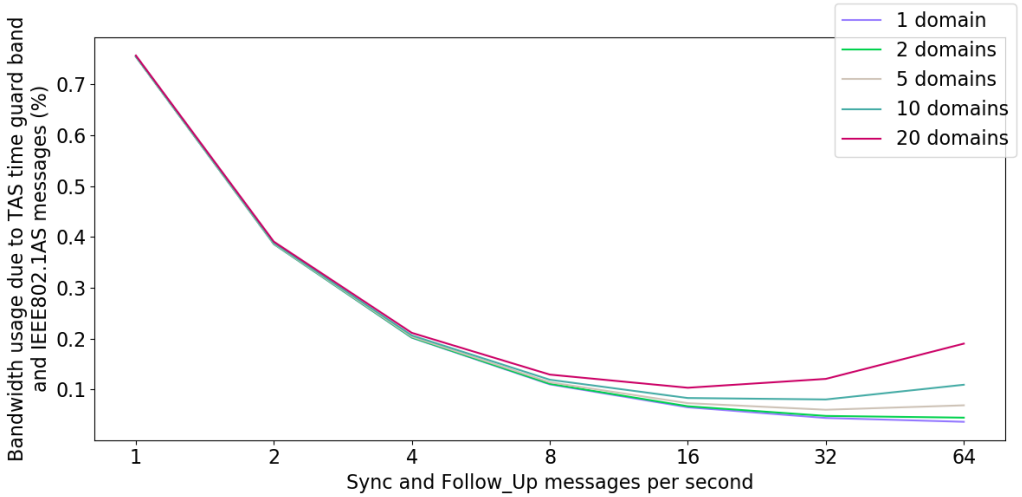


Figure A.26: Bande passante utilisée par les bandes de garde temporelle en fonction de la fréquence de synchronisation avec 184 fenêtres TAS par seconde pour différent nombre de domaines gPTP.

A.4.2 Impact du protocole de synchronisation sur les autres flux de messages

Dans un réseau embarqué critique, un soin tout particulier est porté à la durée de traversée des messages. En effet, il est nécessaire de garantir une latence maximale pour assurer le bon fonctionnement des applications qui repose sur ces messages. L'impact de petits messages, comme ceux du protocole de synchronisation, est généralement très faible. Cependant, la multiplication de ces messages, à cause de leur fréquence ou du nombre de domaines, pourrait avoir un impact. Cette sous-section a pour but de quantifier cet impact.

Cette évaluation repose sur des analyses formelles de pire temps de traversée effectué à l'aide des méthodes de calcul réseau. Pour ce faire, nous utiliserons le logiciel de Timaeus-Net sur le cas d'étude du satellite. Un flux de chaque niveau de priorité a été sélectionné pour illustrer les résultats. Les flux de synchronisation utilisent le niveau de priorité 4.

Figure A.27 illustre l'influence de la fréquence de synchronisation sur la latence pire cas des flux de différent niveau de priorité. On y observe que seul le flux le moins prioritaire est visiblement impacté par l'évolution de la fréquence de synchronisation. En réalité, le flux de priorité 3 est aussi impacté dans un intervalle compris entre -2 ns et +4 ns. On note que, les flux les moins prioritaires subissent une augmentation de leur temps de traversée, pire cas lorsque la fréquence augmente. Cependant, ces flux sont les moins contraints. Les flux de plus haute priorité, les plus contraints, ne sont pas impactés par l'augmentation de fréquence.

Le même constat peut-être réalisé lorsque le nombre de domaines augmente comme le montre la Figure A.28.

A.5 Conclusion

Dans cette annexe, nous avons résumé les travaux présentés dans ce manuscrit où nous cherchions à lever les verrous qui empêche l'utilisation de IEEE802.1AS dans les réseaux embarqué critique. Ces

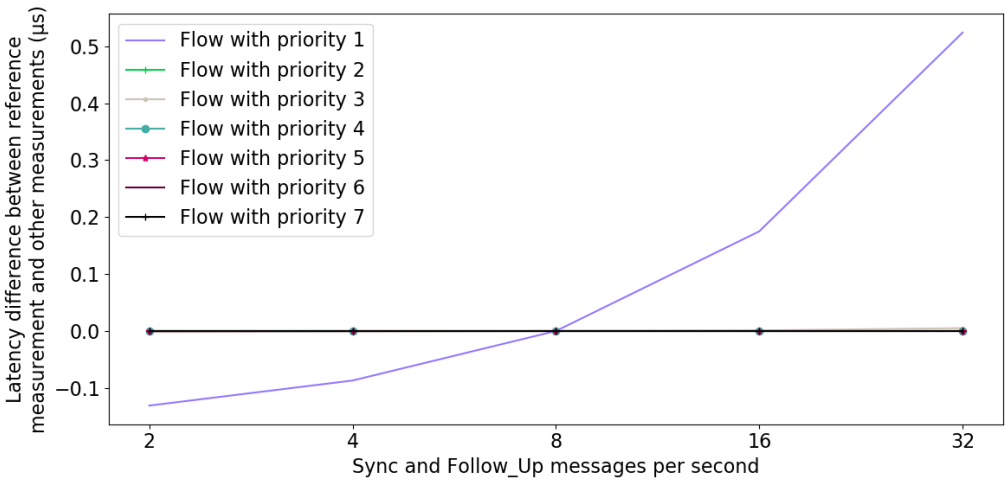


Figure A.27: Impact du nombre de message `Sync` et `Follow_Up` par seconde sur le temps de traversée pire cas de sept flux de différent niveau de priorité.

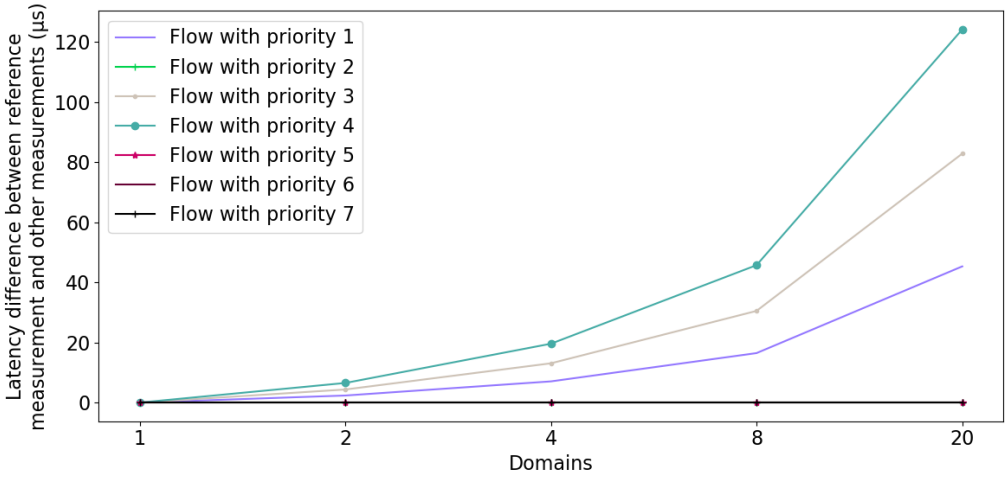


Figure A.28: Impact du nombre de domaines sur l'évolution de la latence de traversée pire cas pour sept flux de différent niveau de priorité

verrous sont liés à la précision et à la robustesse aux pannes du protocole.

Ainsi, dans un premier temps, nous avons étudié la précision atteignable avec le protocole. Pour ce faire, nous avons commencé par rendre une librairie open source représentative de la réalité à l'aide d'un grand nombre de mesures sur du matériel implémentant IEEE802.1AS. De plus, le protocole expérimental de calibration et validation de la librairie est facilement reproductible pour d'autre matériel et couche physique. Nous avons ensuite utilisé ces résultats pour proposer une série d'équations permettant de calculer des bornes supérieures et inférieures sur la précision pire cas. Ces nouvelles équations sont moins pessimiste que l'état de l'art, supportent plusieurs couches physiques et ont été finement comparés à la précision mesuré sur des équipements réels.

Dans un second temps, l'aspect robustesse aux pannes a été étudié. Pour commencer, nous avons comparé les deux mécanismes proposés par le standard pour déterminer lequel était le plus adapté au réseau embarqué critique. Nous sommes arrivés à la conclusion que bien qu'ayant des performances similaires sur des petits réseaux, la complexité d'expliquer le comportement du BMCA ainsi que le manque de borne sur le temps de reconfiguration sont des freins majeurs pour son utilisation. Cependant, la configuration statique souffre d'une complexité importante lors de sa conception. Pour lever cette limitation, nous proposons une méthode qui permet de déterminer la configuration la plus précise parmi les plus robustes. Cette méthode est basée sur l'utilisation de métrique pour évaluer et comparer les configurations entre elle afin d'effectuer une sélection. Cette méthode s'est révélée capable de réduire le vaste ensemble de recherche à une solution en quelques heures sur nos cas d'étude les plus connectés.

Et enfin, nous avons étudié l'influence des paramètres dimensionnant de la précision et de la robustesse sur la consommation de bande passante et sur les latences pires cas des flux applicatif. Nous avons montré que malgré l'augmentation du nombre de messages causé par la fréquence de synchronisation et le nombre de domaines a un impact sur la consommation de bande passante et sur les latences pires cas des flux moins prioritaire, mais cet impact reste relativement faible. De plus, il est prévisible et quantifiable simplement.

Appendix B

Iterative bound computation in Python

```
#Illustration of the iterative calculation of the bound
#on a daisy chain of 10 devices with a poor quality
#clock at hop 8

def iterativeUpperBound(Is, Ip, ppmGm, ppmParent,
                        ppmChild, T, G, A, dmin, Jfup,
                        JParentChild, JChildParent, i,
                        rNoErrorParent, rParent,
                        cNoErrorParent, cParent):
    #neighborRateRatio computation
    nr = (Ip+Ip*ppmChild)/(Ip-Ip*ppmParent)
    dnr = (2*G + G*(ppmParent-ppmChild) +JParentChild*(1+ppmParent) ) / \
        (Ip*(1 - 2*ppmChild + pow(ppmChild,2)) + (ppmParent - 1)*(G +JParentChild))

    #pdelay computation
    dD =  (((T + 2 * dmin +JParentChild + JChildParent + A)*(ppmChild + 1) + G)* \
        (nr + dnr) - (T*(1-ppmParent)-G))/2 - dmin

    #rateRatio computation
    rNoError = rNoErrorParent * nr
    r = rParent * (nr + dnr)
    dr = r - rNoError

    #correctionField computation
    cNoError = cNoErrorParent + dmin*rNoErrorParent + T*rNoError
    c = cParent + (dmin + dD)*rParent + (T+G)*r
    dc = cParent - cNoErrorParent

    #Drift computation
    drift = (abs(ppmGm)+abs(ppmChild))*(Is + Jfup)

    #Precision computation
    p = drift + dc + dD + G

    return p, rNoError, r, cNoError, c

def iterativeLowerBound(Is, Ip, ppmGm, ppmParent,
                        ppmChild, T, G, A, dmin, Jfup,
                        JParentChild, JChildParent, i,
                        rNoErrorParent, rParent,
                        cNoErrorParent, cParent):
    #neighborRateRatio computation
    nr = (Ip-Ip*ppmChild)/(Ip+Ip*ppmParent)
    dnr = (-2*G + -G*(ppmChild-ppmParent) -JParentChild*(1-ppmParent) ) / \
        (Ip*(1+ 2*ppmChild + pow(ppmChild,2)) + (ppmParent+1)*(G +JParentChild))

    #pdelay computation
    dD =  (((T + 2 * dmin + A)*(1-ppmChild) - G)*(nr + dnr)\
        - (T*(ppmParent + 1)+G))/2 - (dmin + JGmL + A)

    #rateRatio computation
    rNoError = rNoErrorParent * nr
    r = rParent * (nr + dnr)
    dr = r - rNoError

    #correctionField computation
    cNoError = cNoErrorParent + dmin*rNoErrorParent + T*rNoError
    c = cParent + (dmin + dD)*rParent + (T-G)*r
    dc = cParent - cNoErrorParent

```

```
61      #Drift computation
62      drift = -(abs(ppmGm)+abs(ppmChild))*(Is + Jfup)
63
64      #Precision computation
65      p = drift + dc + dD - 2*G
66
67      return p, rNoError, r, cNoError, c
68
69
70
71  if __name__=="__main__":
72      ############################ Parameters ######################################
73      ##Protocol
74      Ip = 1                                  #unit : s
75      Is = 0.125                              #unit : s
76      Jfup = 0.002                            #unit : s
77
78      ##Clock
79      ppmArr = [10 * pow(10,-6)]*10           #unit : ppm
80      ppmArr[0] = 0.02 * pow(10,-6)           #unit : ppm     #Grandmaster drift
81      ppmArr[8] = 50*pow(10,-6)               #unit : ppm     #Bad clock in the daisy chain
82
83      G = 10*pow(10,-9)                       #unit : s
84
85      ##Physical layer (1Gb/s)
86      A = (4.95*3 - 8) * pow(10,-9)           #unit : s
87      JGmL = (4.95*3*2) * pow(10,-9)          #unit : s
88      JLGm = 8*pow(10,-9)                     #unit : s
89
90      ##Hardware behaviour
91      T = 1 * pow(10,-3)                      #unit : s
92      dmin = 200 * pow(10,-9)                 #unit : s
93
94      #Initial Values
95      rNoErrorParentUpper, rNoErrorParentLower = 1,1
96      rParentUpper, rParentLower = 1,1
97      cNoErrorParentUpper, cNoErrorParentLower = 0,0
98      cParentUpper, cParentLower = 0, 0
99
100     #Print worst-case precision upper and lower bounds at every hop
101     for i in range(1,10):
102         pUpper, rNoErrorParentUpper, rParentUpper, cNoErrorParentUpper, \
103             cParentUpper = iterativeUpperBound(Is, Ip, ppmArr[0], ppmArr[i-1],
104                 ppmArr[i], T, G, A, dmin, Jfup, JGmL, JLGm, i,
105                 rNoErrorParentUpper, rParentUpper, cNoErrorParentUpper,
106                 cParentUpper)
107         pLower, rNoErrorParentLower, rParentLower, cNoErrorParentLower, \
108             cParentLower = iterativeLowerBound(Is, Ip, ppmArr[0], ppmArr[i-1],
109                 ppmArr[i], T, G, A, dmin, Jfup, JGmL, JLGm, i,
110                 rNoErrorParentLower, rParentLower, cNoErrorParentLower,
111                 cParentLower)
112         print(f"Hop {i}: \n\tUpper bound : {pUpper*pow(10,6):.3f}us \
113             \n\tLower bound : {pLower*pow(10,6):.3f}us")
```

Appendix C

3-spanning tree set robustness score

Spanning tree set	Robustness score	Rank
(0, 1, 2)	7767	43
(0, 1, 3)	7767	43
(0, 1, 4)	7863	49
(0, 1, 5)	7857	48
(0, 1, 6)	7863	49
(0, 1, 7)	7431	25
(0, 1, 8)	7591	36
(0, 1, 9)	7585	35
(0, 1, 10)	7591	36
(0, 1, 11)	7159	11
(0, 2, 3)	7767	43
(0, 2, 4)	7541	29
(0, 2, 5)	7431	25
(0, 2, 6)	7541	29
(0, 2, 7)	7517	27
(0, 2, 8)	7269	18
(0, 2, 9)	7159	11
(0, 2, 10)	7269	18
(0, 2, 11)	7245	15
(0, 3, 4)	7431	25
(0, 3, 5)	7425	24
(0, 3, 6)	7431	25
(0, 3, 7)	7407	22
(0, 3, 8)	7159	11
(0, 3, 9)	7153	10
(0, 3, 10)	7159	11
(0, 3, 11)	7135	7
(0, 4, 5)	7863	49
(0, 4, 6)	7955	52
(0, 4, 7)	7523	28
(0, 4, 8)	7701	42
(0, 4, 9)	7591	36
(0, 4, 10)	7683	40
(0, 4, 11)	7251	16
(0, 5, 6)	7845	47
(0, 5, 7)	7413	23
(0, 5, 8)	7591	36
(0, 5, 9)	7591	36
(0, 5, 10)	7573	33
(0, 5, 11)	7141	8
(0, 6, 7)	7541	29
(0, 6, 8)	7683	40
(0, 6, 9)	7573	33
(0, 6, 10)	7701	42
(0, 6, 11)	7269	18

Spanning tree set	Robustness score	Rank
(0, 7, 8)	7251	16
(0, 7, 9)	7141	8
(0, 7, 10)	7269	18
(0, 7, 11)	7269	18
(0, 8, 9)	7591	36
(0, 8, 10)	7668	38
(0, 8, 11)	7236	14
(0, 9, 10)	7558	31
(0, 9, 11)	7126	6
(0, 10, 11)	7269	18
(1, 2, 3)	7767	43
(1, 2, 4)	7431	25
(1, 2, 5)	7425	24
(1, 2, 6)	7431	25
(1, 2, 7)	7407	22
(1, 2, 8)	7159	11
(1, 2, 9)	7153	10
(1, 2, 10)	7159	11
(1, 2, 11)	7135	7
(1, 3, 4)	7431	25
(1, 3, 5)	7677	39
(1, 3, 6)	7431	25
(1, 3, 7)	7407	22
(1, 3, 8)	7159	11
(1, 3, 9)	7405	21
(1, 3, 10)	7159	11
(1, 3, 11)	7135	7
(1, 4, 5)	7857	48
(1, 4, 6)	7845	47
(1, 4, 7)	7413	23
(1, 4, 8)	7591	36
(1, 4, 9)	7585	35
(1, 4, 10)	7573	33
(1, 4, 11)	7141	8
(1, 5, 6)	7839	46
(1, 5, 7)	7407	22
(1, 5, 8)	7585	35
(1, 5, 9)	7837	45
(1, 5, 10)	7567	32
(1, 5, 11)	7135	7
(1, 6, 7)	7431	25
(1, 6, 8)	7573	33
(1, 6, 9)	7567	32
(1, 6, 10)	7591	36
(1, 6, 11)	7159	11

Spanning tree set	Robustness score	Rank
(1, 7, 8)	7141	8
(1, 7, 9)	7135	7
(1, 7, 10)	7159	11
(1, 7, 11)	7159	11
(1, 8, 9)	7585	35
(1, 8, 10)	7558	31
(1, 8, 11)	7126	6
(1, 9, 10)	7552	30
(1, 9, 11)	7120	5
(1, 10, 11)	7159	11
(2, 3, 4)	7431	25
(2, 3, 5)	7425	24
(2, 3, 6)	7431	25
(2, 3, 7)	8399	56
(2, 3, 8)	7159	11
(2, 3, 9)	7153	10
(2, 3, 10)	7159	11
(2, 3, 11)	8127	53
(2, 4, 5)	7431	25
(2, 4, 6)	7523	28
(2, 4, 7)	7499	26
(2, 4, 8)	7269	18
(2, 4, 9)	7159	11
(2, 4, 10)	7251	16
(2, 4, 11)	7227	13
(2, 5, 6)	7413	23
(2, 5, 7)	7389	20
(2, 5, 8)	7159	11
(2, 5, 9)	7159	11
(2, 5, 10)	7141	8
(2, 5, 11)	7117	4
(2, 6, 7)	7517	27
(2, 6, 8)	7251	16
(2, 6, 9)	7141	8
(2, 6, 10)	7269	18
(2, 6, 11)	7245	15
(2, 7, 8)	7227	13
(2, 7, 9)	7117	4
(2, 7, 10)	7245	15
(2, 7, 11)	8237	55
(2, 8, 9)	7159	11
(2, 8, 10)	7236	14
(2, 8, 11)	7212	12
(2, 9, 10)	7126	6
(2, 9, 11)	7102	2

Spanning tree set	Robustness score	Rank
(2, 10, 11)	7245	15
(3, 4, 5)	7425	24
(3, 4, 6)	7413	23
(3, 4, 7)	7389	20
(3, 4, 8)	7159	11
(3, 4, 9)	7153	10
(3, 4, 10)	7141	8
(3, 4, 11)	7117	4
(3, 5, 6)	7407	22
(3, 5, 7)	7383	19
(3, 5, 8)	7153	10
(3, 5, 9)	7405	21
(3, 5, 10)	7135	7
(3, 5, 11)	7111	3
(3, 6, 7)	7407	22
(3, 6, 8)	7141	8
(3, 6, 9)	7135	7
(3, 6, 10)	7159	11
(3, 6, 11)	7135	7
(3, 7, 8)	7117	4
(3, 7, 9)	7111	3
(3, 7, 10)	7135	7
(3, 7, 11)	8127	53
(3, 8, 9)	7153	10
(3, 8, 10)	7126	6
(3, 8, 11)	7102	2
(3, 9, 10)	7120	5
(3, 9, 11)	7096	1
(3, 10, 11)	7135	7
(4, 5, 6)	7863	49
(4, 5, 7)	7431	25
(4, 5, 8)	7927	51
(4, 5, 9)	7927	51
(4, 5, 10)	7591	36
(4, 5, 11)	7159	11
(4, 6, 7)	7541	29
(4, 6, 8)	7701	42
(4, 6, 9)	7591	36
(4, 6, 10)	7701	42
(4, 6, 11)	7269	18
(4, 7, 8)	7269	18
(4, 7, 9)	7159	11
(4, 7, 10)	7269	18
(4, 7, 11)	7269	18
(4, 8, 9)	7927	51

Spanning tree set	Robustness score	Rank
(4, 8, 10)	7686	41
(4, 8, 11)	7254	17
(4, 9, 10)	7576	34
(4, 9, 11)	7144	9
(4, 10, 11)	7269	18
(5, 6, 7)	7431	25
(5, 6, 8)	7591	36
(5, 6, 9)	7591	36
(5, 6, 10)	7591	36
(5, 6, 11)	7159	11
(5, 7, 8)	7159	11
(5, 7, 9)	7159	11
(5, 7, 10)	7159	11
(5, 7, 11)	7159	11
(5, 8, 9)	7927	51
(5, 8, 10)	7576	34
(5, 8, 11)	7144	9
(5, 9, 10)	7576	34
(5, 9, 11)	7144	9
(5, 10, 11)	7159	11
(6, 7, 8)	7269	18
(6, 7, 9)	7159	11
(6, 7, 10)	7605	37
(6, 7, 11)	7605	37
(6, 8, 9)	7591	36
(6, 8, 10)	7686	41
(6, 8, 11)	7254	17
(6, 9, 10)	7576	34
(6, 9, 11)	7144	9
(6, 10, 11)	7605	37
(7, 8, 9)	7159	11
(7, 8, 10)	7254	17
(7, 8, 11)	7254	17
(7, 9, 10)	7144	9
(7, 9, 11)	7144	9
(7, 10, 11)	7605	37
(8, 9, 10)	8200	54
(8, 9, 11)	7768	44
(8, 10, 11)	7878	50
(9, 10, 11)	7768	44

Table C.1: Score and ranking obtained by the 220 3-spanning tree set rooted in 0 of the automotive topology with the robustness metric

Appendix D

List of publications

- ”Assessing a precise gPTP simulator with IEEE802.1AS hardware measurements” in ERTS 2022
- ”TSN Network example with a Hot-Standby Grandmaster for critical embedded applications” at TSN/A 2022
- ”Worst-case synchronization precision of IEEE802.1AS” in ETFA 2023
- ”Designing reliable configuration for TSN synchronization on automotive networks” at E&IP@ATD 2023

Bibliography

[1] Mil-std-1553: Aircraft internal time division command/response multiplex data bus. 1975.

[2] Daytime Protocol. RFC 867, May 1983. URL: https://www.rfc-editor.org/info/rfc867, doi:10.17487/RFC0867.

[3] Time Protocol. RFC 868, May 1983. URL: https://www.rfc-editor.org/info/rfc868, doi:10.17487/RFC0868.

[4] Algorithms for synchronizing network clocks. RFC 956, September 1985. URL: https://www.rfc-editor.org/info/rfc956, doi:10.17487/RFC0956.

[5] Network Time Protocol (NTP). RFC 958, September 1985. URL: https://www.rfc-editor.org/info/rfc958, doi:10.17487/RFC0958.

[6] Network Time Protocol (version 1) specification and implementation. RFC 1059, July 1988. URL: https://www.rfc-editor.org/info/rfc1059, doi:10.17487/RFC1059.

[7] Network Time Protocol (version 2) specification and implementation. RFC 1119, September 1989. URL: https://www.rfc-editor.org/info/rfc1119, doi:10.17487/RFC1119.

[8] Standard for local and metropolitan area networks: Media access control (mac) bridges. *IEEE Std 802.1D-1990*, pages 1–176, 1991. doi:10.1109/IEEESTD.1991.101050.

[9] Network Time Protocol (Version 3) Specification, Implementation and Analysis. RFC 1305, March 1992. URL: https://www.rfc-editor.org/info/rfc1305, doi:10.17487/RFC1305.

[10] Iso 11898 : Road vehicles — interchange of digital information — controller area network (can) for high-speed communication. 1993.

[11] Do-254: Design assurance guidance for airborne electronic hardware. 2000.

[12] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2002*, pages 1–154, 2002. doi:10.1109/IEEESTD.2002.94144.

[13] Ieee standards for local and metropolitan area networks - amendment to 802.1q virtual bridged local area networks: Multiple spanning trees. *IEEE Std 802.1s-2002 (Amendment to IEEE Std 802.1Q, 1998 Edition)*, pages 1–211, 2002. doi:10.1109/IEEESTD.2002.94223.

[14] Ieee standard for local and metropolitan area networks: Media access control (mac) bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 1–281, 2004. doi:10.1109/IEEESTD.2004.94569.

[15] Arinc 664 p7. 2005.

[16] G.8261 : Timing and synchronization aspects in packet networks. 2006.

[17] Ecss-e-st-50-12c – spacewire – links, nodes, routers and networks. 2008.

[18] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pages 1–269, 2008. doi:10.1109/IEEESTD.2008.4579760.

[19] Ieee standard for local and metropolitan area networks– virtual bridged local area networks amendment 12: Forwarding and queuing enhancements for time-sensitive streams. *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)*, pages 1–72, 2010. doi:10.1109/IEEESTD.2010.8684664.

[20] Ieee standard for local and metropolitan area networks–virtual bridged local area networks amendment 14: Stream reservation protocol (srp). *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)*, pages 1–119, 2010. doi:10.1109/IEEESTD.2010.5594972.

[21] Ieee standard for layer 2 transport protocol for time sensitive applications in a bridged local area network. *IEEE Std 1722-2011*, pages 1–65, 2011. doi:10.1109/IEEESTD.2011.5764875.

[22] Ieee standard for local and metropolitan area networks - timing and synchronization for time-sensitive applications in bridged local area networks. *IEEE Std 802.1AS-2011*, pages 1–292, 2011. doi:10.1109/IEEESTD.2011.5741898.

[23] Do-178c: Software considerations in airborne systems and equipment certification. 2012.

[24] Iso 17458-1: Road vehicles — flexray communications system — part 1: General information and use case definition. 2013.

[25] Iec 62439: High availability automation networks - part 3: Parallel redundancy protocol (prp) and high-availability seamless redundancy (hsr). *INTERNATIONAL ELECTROTECHNICAL COMMISSION*, 2014.

[26] Ieee standard for local and metropolitan area networks – bridges and bridged networks – amendment 26: Frame preemption. *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)*, pages 1–52, 2016. doi:10.1109/IEEESTD.2016.7553415.

[27] Ieee standard for local and metropolitan area networks – bridges and bridged networks - amendment 25: Enhancements for scheduled traffic. *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)*, pages 1–57, 2016. doi:10.1109/IEEESTD.2016.8613095.

[28] Iso 17987-1: Road vehicles — local interconnect network (lin) — part 1: General information and use case definition. 2016.

[29] Ieee standard for local and metropolitan area networks–bridges and bridged networks– amendment 28: Per-stream filtering and policing. *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)*, pages 1–65, 2017. doi:10.1109/IEEESTD.2017.8064221.

[30] Ieee standard for local and metropolitan area networks–frame replication and elimination for reliability. *IEEE Std 802.1CB-2017*, pages 1–102, 2017. doi:10.1109/IEEESTD.2017.8091139.

[31] Ieee standard for local and metropolitan area networks-media access control (mac) security. *IEEE Std 802.1AE-2018 (Revision of IEEE Std 802.1AE-2006)*, pages 1–239, 2018. doi:10.1109/IEEESTD.2018.8585421.

[32] Ieee standard for a precision clock synchronization protocol for networked measurement and control systems. *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)*, pages 1–499, 2020. doi:10.1109/IEEESTD.2020.9120376.

[33] Ieee standard for local and metropolitan area networks–bridges and bridged networks - amendment 34:asynchronous traffic shaping. *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)*, pages 1–151, 2020. doi:10.1109/IEEESTD.2020.9253013.

[34] Ieee standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications. *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)*, pages 1–421, 2020. doi:10.1109/IEEESTD.2020.9121845.

[35] Iso 21806-1: Road vehicles — media oriented systems transport (most) — part 1: General information and definitions. 2020.

[36] Inet framework, 2023. URL: https://inet.omnetpp.org/.

[37] Omnet++ simulator, 2023. URL: https://omnetpp.org/.

[38] Shimmi Asokan and G. Kumar. Formal modeling of the gptp clock synchronization algorithm in automotive ethernet. *Innovations in Systems and Software Engineering*, pages 1–17, 09 2022. doi:10.1007/s11334-022-00483-1.

[39] Quentin Bailleul, Katia Jaffrès-Runser, Jean-Luc Scharbarg, and Philippe Cuenot. Assessing a precise gptp simulator with ieee802.1as hardware measurements. In *11th European Congress on Embedded Real-Time Systems (ERTS)*, 2022.

[40] Quentin Bailleul, Katia Jaffrès-Runser, Jean-Luc Scharbarg, and Philippe Cuenot. Tsn network example with a hot-standby grandmaster for critical embedded applications. In *TSN/A (TSN/A)*, 2022.

[41] Quentin Bailleul, Katia Jaffrès-Runser, Jean-Luc Scharbarg, and Philippe Cuenot. Worst-case synchronization precision of ieee802.1as. In *28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2023.

[42] Quentin Bailleul, Katia Jaffrès-Runser, Jean-Luc Scharbarg, Daniel Hopf, and Philippe Cuenot. Designing reliable configuration for tsn synchronization on automotive networks. In *Ethernet & IP @ Automotive Technology Day (E&IP@ATD)*, 2023.

[43] Haytham Baniabdelghany, Roman Obermaisser, and Ala' Khalifeh. Extended synchronization protocol based on ieee802.1as for improved precision in dynamic and asymmetric tsn hybrid networks. In *2020 9th Mediterranean Conference on Embedded Computing (MECO)*, pages 1–8, 2020. doi:10.1109/MECO49872.2020.9134100.

[44] Andy Bierman, Martin Björklund, and Kent Watsen. RESTCONF Protocol. RFC 8040, January 2017. URL: https://www.rfc-editor.org/info/rfc8040, doi:10.17487/RFC8040.

[45] Alessio Buscemi, Manasvi Ponaka, Mahdi Fotouhi, Christian Koebel, Florian Jomrich, and Thomas Engel. An intrusion detection system against rogue master attacks on gptp. In *IEEE Vehicular Technology Conference (VTC2023-Spring), Florence 20-23 June 2023*, 2023.

[46] Pierre-Julien Chaine, Marc Boyer, Claire Pagetti, and Franck Wartel. TSN support for quality of service in space. In *10th European Congress on Embedded Real Time Software and Systems (ERTS 2020)*, 2020.

[47] Benoit Claise, Joe Clarke, and Jan Lindblad. *Network Programmability with YANG*. Addison-Wesley, 2019.

[48] Hugo Daigmorte and Marc Boyer. Traversal time for weakly synchronized can bus. In *Proceedings of the 24th International Conference on Real-Time Networks and Systems*, pages 35–44, 2016.

[49] Marcelo Dalmas, Higor Rachadel, Gustavo Silvano, and Carlos Dutra. Improving ptp robustness to the byzantine failure. In *2015 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 111–114, 2015. doi:10.1109/ISPCS.2015.7324693.

[50] Robert I Davis, Alan Burns, Reinder J Bril, and Johan J Lukkien. Controller area network (can) schedulability analysis: Refuted, revisited and revised. *Real-Time Systems*, 35:239–272, 2007.

[51] Aboubacar Diarra, Thomas Hogenmueller, Armin Zimmermann, Andreas Grzemba, and Umair Asrar Khan. Improved clock synchronization start-up time for switched ethernet-based in-vehicle networks.

[52] Djamel Djenouri and Miloud Bagaa. Synchronization protocols and implementation issues in wireless sensor networks: A review. *IEEE Systems Journal*, 10(2):617–627, 2014.

[53] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *SIGOPS Oper. Syst. Rev.*, 36(SI):147–163, dec 2003. doi:10.1145/844128.844143.

[54] Darin England, Bharadwaj Veeravalli, and Jon B. Weissman. A robust spanning tree topology for data collection and dissemination in distributed environments. *IEEE Transactions on Parallel and Distributed Systems*, 18(5):608–620, 2007. doi:10.1109/TPDS.2007.1032.

[55] Rob Enns, Martin Björklund, Andy Bierman, and Jürgen Schönwälder. Network Configuration Protocol (NETCONF). RFC 6241, June 2011. URL: https://www.rfc-editor.org/info/rfc6241, doi:10.17487/RFC6241.

[56] AS-2D2 Deterministic Ethernet and Unified Networking. *Time-Triggered Ethernet*, 2011. URL: https://doi.org/10.4271/AS6802, doi:https://doi.org/10.4271/AS6802.

[57] P. Ferrari, A. Flammini, S. Rinaldi, A. Bondavalli, and F. Brancati. Improving robustness of the synchronization quality of ieee1588 nodes. In *2010 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 36–41, 2010. doi:10.1109/ISPCS.2010.5609784.

[58] Mahdi Fotouhi, Alessio Buscemi, Abdelwahab Boualouache, Florian Jomrich, Christian Koebel, and Thomas Engel. Assessing the impact of attacks on an automotive ethernet time synchronization testbed. In *2023 IEEE Vehicular Networking Conference (VNC)*, pages 223–230, 2023. doi:10.1109/VNC57357.2023.10136275.

[59] P. Fragopoulou and S.G. Akl. Edge-disjoint spanning trees on the star network with applications to fault tolerance. *IEEE Transactions on Computers*, 45(2):174–185, 1996. doi:10.1109/12.485370.

[60] Georg Gaderer, Albert Treytl, and Thilo Sauter. Security aspects for ieee 1588 based clock synchronization protocols. In *Proc. IEEE Int. Workshop Factory Commun. Syst.(WFCS)*, pages 247–250. Citeseer, 2006.

[61] Geoffrey M. Garner, Aaron Gelter, and Michael Johas Teener. New simulation and test results for ieee 802.1as timing performance. In *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–7, 2009. doi:10.1109/ISPCS.2009.5340214.

[62] Marina Gutiérrez, Wilfried Steiner, Radu Dobrin, and Sasikumar Punnekkat. Synchronization quality of IEEE 802.1 AS in large-scale industrial automation networks. In *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, pages 273–282. IEEE, 2017.

[63] Koki Horita, Shota Shiobara, Takao Okamawari, Fumio Teraoka, and Kunitake Kaneko. Ptp accuracy measurement comparison between boundary clock and vlan priority. In *2017 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6, 2017. doi:10.1109/ISPCS.2017.8056748.

[64] Oana Andreea Hotescu, Katia Jaffrès-Runser, Jean-Luc Scharbarg, and Christian Fraboul. Towards Quality of Service Provision with Avionics Full Duplex Switching. In *29th Euromicro Conference on Real-Time Systems (ECRTS 2017)*, pages 1–3, Dubrovnik, Croatia, June 2017. URL: https://hal.science/hal-01919072.

[65] Sun-Yuan Hsieh and Chang-Jen Tu. Constructing edge-disjoint spanning trees in locally twisted cubes. *Theoretical Computer Science*, 410(8-10):926–932, 2009.

[66] Michael D. Johas Teener and Geoffrey M. Garner. Overview and timing performance of ieee 802.1as. In *2008 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 49–53, 2008. doi:10.1109/ISPCS.2008.4659212.

[67] Robert M. Kaminsky. Test results of ieee 1588v2 network synchronization holdover performance using various types of reference oscillators. In *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–5, 2019. doi:10.1109/ISPCS.2019.8886634.

[68] Eleftherios Kyriakakis, Koen Tange, Niklas Reusch, Eder Ollora Zaballa, Xenofon Fafoutis, Martin Schoeberl, and Nicola Dragoni. Fault-tolerant clock synchronization using precise time protocol multi-domain aggregation. In *2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC)*, pages 114–122, 2021. doi:10.1109/ISORC52013.2021.00025.

[69] Jean-Yves Le Boudec. *Performance evaluation of computer and communication systems*. Epfl Press, 2010.

[70] David B. Leeson. Oscillator phase noise: A 50-year review. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 63(8):1208–1225, 2016. doi:10.1109/TUFFC.2016.2562663.

[71] Xiaoting Li, Jean-Luc Scharbarg, and Christian Fraboul. Improving end-to-end delay upper bounds on an afdx network by integrating offsets in worst-case analysis. In *2010 IEEE 15th Conference on Emerging Technologies & Factory Automation (ETFA 2010)*, pages 1–8. IEEE, 2010.

[72] Hyung-Taek Lim, Daniel Herrscher, Lars Völker, and Martin Johannes Waltl. Ieee 802.1as time synchronization in a switched ethernet based in-car network. In *2011 IEEE Vehicular Networking Conference (VNC)*, pages 147–154, 2011. doi:10.1109/VNC.2011.6117136.

[73] Patrick Loschmidt. *On enhanced clock synchronization performance through dedicated ethernet hardware support*. PhD thesis, 2010.

[74] Patrick Loschmidt, Reinhard Exel, and Georg Gaderer. Highly accurate timestamping for ethernet-based clock synchronization. *Journal of Computer Networks and Communications*, 2012, 2012.

[75] Jim Martin, Jack Burbank, William Kasch, and Professor David L. Mills. Network Time Protocol Version 4: Protocol and Algorithms Specification. RFC 5905, June 2010. URL: https://www.rfc-editor.org/info/rfc5905, doi:10.17487/RFC5905.

[76] INTERNATIONAL ORGANIZATION OF LEGAL METROLOGY. *OIML V 2-200*. 2007.

[77] Jörn Migge, Josetxo Villanueva, Nicolas Navet, and Marc Boyer. Insights on the Performance and Configuration of AVB and TSN in Automotive Ethernet Networks. In *9th European Congress on Embedded Real Time Software and Systems (ERTS 2018)*, Toulouse, France, January 2018. URL: https://hal.science/hal-01746132.

[78] Professor David L. Mills. Simple Network Time Protocol (SNTP). RFC 1361, August 1992. URL: https://www.rfc-editor.org/info/rfc1361, doi:10.17487/RFC1361.

[79] Faten Mkacher. *Optimization of Time Synchronization Techniques on Computer Networks*. PhD thesis, 2020. Thèse de doctorat dirigée par Duda, Andrzej Informatique Université Grenoble Alpes 2020. URL: http://www.theses.fr/2020GRALM015.

[80] Andreas Puhm, Aneeq Mahmood, Thomas Bigler, and Nikolaus Kerö. Synchronizing an ieee 1588 slave clock over both paths of a redundant ethernet system. In *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6, 2016. doi:10.1109/ISPCS.2016.7579507.

[81] Henning Puttnies, Peter Danielis, Enkhtuvshin Janchivnyambuu, and Dirk Timmermann. A simulation model of ieee 802.1 as gptp for clock synchronization in omnet++. In *OMNeT++*, pages 63–72, 2018.

[82] William J Riley and David A Howe. Handbook of frequency stability analysis. 2008.

[83] Naotaka Sakaguchi, Ryusuke Kawate, and Yukimasa Nagai. Ieee 802.1as precision time protocol full hardware prototyping for industrial iot. In *2023 IEEE 20th Consumer Communications & Networking Conference (CCNC)*, pages 943–944, 2023. doi:10.1109/CCNC51644.2023.10059981.

[84] Javier Serrano, M Lipinski, T Wlostowski, E Gousiou, Erik van der Bij, M Cattin, and G Daniluk. The white rabbit project. 2013.

[85] Tobias Striffler and Hans D. Schotten. The 5g transparent clock: Synchronization errors in integrated 5g-tsn industrial networks. In *2021 IEEE 19th International Conference on Industrial Informatics (INDIN)*, pages 1–6, 2021. doi:10.1109/INDIN45523.2021.9557468.

[86] Shyue-Ming Tang, Yue-Li Wang, and Yung-Ho Leu. Optimal independent spanning trees on hypercubes. *J. Inf. Sci. Eng.*, 20(1):143–156, 2004.

[87] Albert Treytl and Bernd Hirschler. Security flaws and workarounds for ieee 1588 (transparent) clocks. In *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, pages 1–6. IEEE, 2009.

[88] Jeanette Tsang and Konstantin Beznosov. A security analysis of the precise time protocol (short paper). In *Information and Communications Security: 8th International Conference, ICICS 2006, Raleigh, NC, USA, December 4-7, 2006. Proceedings 8*, pages 50–59. Springer, 2006.

[89] Iñaki Val, Óscar Seijo, Raul Torrego, and Armando Astarloa. Ieee 802.1as clock synchronization performance evaluation of an integrated wired–wireless tsn architecture. *IEEE Transactions on Industrial Informatics*, 18(5):2986–2999, 2022. doi:10.1109/TII.2021.3106568.

[90] Wolfgang Wallner, Armin Wasicek, and Radu Grosu. A simulation framework for ieee 1588. In *2016 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, pages 1–6, 2016. doi:10.1109/ISPCS.2016.7579516.

[91] Luxi Zhao, Paul Pop, Zhong Zheng, Hugo Daigmorte, and Marc Boyer. Latency analysis of multiple classes of avb traffic in tsn with standard credit behavior using network calculus. *IEEE Transactions on Industrial Electronics*, 68(10):10291–10302, 2020.