End-to-End Service Guarantee for High-Speed Optical Networks

Guillaume Soudais

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2024IPPAT027

Thèse de doctorat

End-to-end service guarantee for high-speed Optical networks

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Telecom Paris

École doctorale n°626 Ecole Doctorale de l'Institut Polytechnique de Paris
(ED IP Paris)
Spécialité de doctorat : information, communications, électronique

Thèse présentée et soutenue à Palaiseau, le 28 Octobre 2024, par

GUILLAUME SOUDAIS

Composition du Jury :

Catherine Lepers Professeur, Telecom SudParis	Présidente / Examinatrice
Ramon Casellas Regi Senior Researcher, Centre Tecnològic de Telecomunicacions de Catalunya	Rapporteur
Nicola Calabretta Professor, Eindhoven University of Technology	Rapporteur
Emmanuel Boutillon Professeur, Université Bretagne-Sud	Examinateur
Sébastien Bigo Senior Researcher, Nokia Bell Labs	Examinateur
Yves Mathieu Professeur, Telecom Paris	Directeur de thèse

Acknowledgments

I would like to express my gratitude towards all the people who have guided and supported me during the realization and the difficult process of redaction of my thesis. I extend my thanks to the members of the jury and my rapporteurs, Nicola Calabretta and Ramon Casellas Regi, for having the patience to read my manuscript and providing me with insightful questions during the defense.

I want to thank Nihel Benzaoui and Mijail Szczerban for introducing me to the field during my internship at Nokia. Their passion for this field guided me towards pursuing a PhD along them.

Then I want to thank Yves Mathieu and Tarik Graba from Télécom Paris. With being my PhD director and supervisors from Télécom, they were my teachers in my first year of engineering school and contributed to my affection for the experimental platform I use in my PhD. I want to thank them for their continuous support during and after my three years of PhD.

I want to thank Sébastien Bigo who did a lot more than supervise my work at Nokia during those three years of PhD. I greatly appreciated our conversations about restaurants and cinema, and I continue to enjoy reading your blog about the Cannes Film Festival.

I am grateful to Nokia Bell Labs and the wonderful team that surrounded me during this journey. You all made the workplace enjoyable, even during the challenging times of COVID. I was glad to be a part of that team.

Finally, I would like to thank my friends and family for their constant encouragement. A special thanks to Alix and Aymeric, whom I met at Nokia, for making tea breaks and climbing sessions after work such memorable parts of this journey.

Abstract

Driven by an ever-growing bandwidth and performance need, the IT network has grown such that OT and telecommunications networks are looking to exploit this infrastructure for their expansion. These three sectors have historically been separated due to different requirements, on latency, its variation and on reliability.

To answer to time critical application needs, the Time Sensitive Network taskforce has developed new sets of protocols for Ethernet networks which are starting to be implemented in commercial products. Other groups have proposed novel architecture with time control to enable guaranteed performances between and inside edge datacenters.

In my PhD I propose a solution to carry time critical application in legacy networks as it does not require changing the whole architecture. I show the benefits of its implementation in TSN networks for a future-proof solution with improved resource usage. To carry time critical traffic in legacy networks, I propose to create a path by isolating and scheduling time critical traffic on a channel with guaranteed latency. With this construction, I build an algorithm to perform latency variation compensation enabling a constant latency transmission for time critical traffic. In a second time, I propose a synchronization scheme and implemented a monitoring network primarily used here for latency monitoring, helping me gain insights on the distribution of latency that my protocol creates. Lastly, with an improved latency compensation algorithm, I demonstrate better jitter performances and study the turn-up time for our protocol enabling resource usage only when time-critical traffic is present.

In my PhD we demonstrate, with an FPGA implementation and commercial product, latency variation reduction enabling OT and telecommunications network applications to run on legacy and TSN augmented network.

Résumé

La crise du COVID a mis en évidence l'importance cruciale des infrastructures numériques, non seulement pour l'économie mondiale, mais aussi pour les interactions sociales. La mission principale de cette infrastructure numérique est de transporter des données entre deux points finaux le long d'un chemin, le plus souvent en utilisant la lumière. Pour des raisons économiques, chaque paire de points finaux ne dispose pas d'un chemin dédié pour échanger son flux de données. Au contraire, une grande partie de ce chemin est partagée avec d'autres flux. Plus précisément, les données provenant de différentes sources ou destinations sont agrégées (multiplexées de manière passive et/ou commutées dynamiquement) en flux de données plus importants et voyagent ensemble à travers des nœuds. Elles sont désagrégées dans d'autres nœuds plus loin, jusqu'à atteindre leurs destinations. Les terminaux, les nœuds, le réseau de chemins et l'intelligence permettant de le reconfigurer sont généralement ce qu'on appelle le « réseau de communication ».

Bien que ce réseau de communication semble être un réseau mondial unique, il est plus exact de le décrire comme une concaténation de différents réseaux, chacun optimisé pour les performances et les coûts selon des exigences spécifiques. En particulier, trois segments de réseau se sont développés au fil du temps, presque de manière isolée, guidés par des besoins particuliers et correspondant à des attentes de marché spécifiques. Naturellement, ces trois segments ont été construits à partir de technologies de réseau tout aussi spécifiques.

1. Le segment de réseau OT (technologie d'exploitation) a été développé pour les environnements fermés des usines, afin de connecter, gérer et surveiller les différentes machines et capteurs présents sur les chaînes de production. Ces réseaux ont été conçus pour garantir la sécurité et la fiabilité, en assurant la protection des travailleurs et en minimisant les temps d'arrêt. Ces exigences, combinées à l'évolution lente des besoins dans les environnements industriels, ont rendu ces réseaux rigides.

2. Le segment de réseau IT (technologies de l'information) a été conçu pour faciliter les échanges de données entre ordinateurs et serveurs. Il a été optimisé pour offrir une flexibilité maximale, répondant ainsi aux schémas typiques des applications informatiques où les points finaux changent fréquemment et sur des durées de vie très courtes (par exemple, souvent inférieures à une milliseconde). Ces réseaux peuvent connecter des points finaux à l'échelle mondiale. Les protocoles Ethernet et Internet, largement utilisés dans les réseaux IT, ont été développés pour permettre le routage des paquets, créant un réseau dynamique où les points finaux peuvent modifier leur emplacement. Cependant, ces protocoles adoptent une approche dite "au mieux" (*best-effort*) pour accéder aux canaux de communication, ce qui limite leur fiabilité.

3. Le segment des réseaux de télécommunications a été initialement développé pour transmettre des messages, puis des communications vocales sur de longues distances. La transmission de la voix sur ces réseaux a introduit la nécessité de réserver un chemin complet pour permettre une reconstruction correcte de la communication vocale dans son intégralité. Ces exigences ont donné naissance à un réseau peu flexible et difficile à faire évoluer à mesure que les débits de données augmentaient. De ces réseaux, nous avons hérité des réseaux de communication sans fil, qui permettent de connecter nos téléphones à Internet. Ces réseaux ont évolué à travers différentes phases et générations, jusqu'à l'arrivée de la 5G, avec l'ambition de fusionner le réseau de télécommunications avec les réseaux IT.

Parmi ces trois segments de réseau, le réseau IT a connu la plus forte augmentation de trafic, atteignant une bande passante de 1 217 Tbps en 2023, principalement dominée par le trafic audio et vidéo, tout en prenant également en charge des applications plus sécurisées, comme la banque en ligne.

Avec l'introduction de la 5G, les réseaux de télécommunications deviennent plus flexibles, et le traitement ainsi que la collecte de données dans les usines sont de plus en plus externalisés hors de leurs locaux. Dans ce contexte, l'infrastructure des réseaux IT est considérée comme un support adéquat pour ces applications.

Cependant, les réseaux de télécommunications et OT imposent des contraintes que les réseaux IT n'ont pas été conçus pour satisfaire. Il y a trois contraintes principales sur les applications, que l'on appelle *temps-critique*, communiquant à travers ces réseaux.

1. *Latence*: Certaines applications exigent que les informations soient transmises entre différents points finaux dans un délai strictement limité.

2. *Fiabilité*: Certaines applications peuvent tolérer une perte d'informations limitée et prédéfinie au cours de la communication.

3. *Gigue*: Certaines applications nécessitent une latence constante ou une variation limitée dans un intervalle prédéfini.

Lors de la construction du réseau IT, l'accent a été mis sur la dynamique et la flexibilité du réseau. Le réseau a été conçu pour s'adapter à des flux de trafic courts (~1 ms). L'accès au canal de communication se fait selon une approche "au mieux" (best-effort), où les paquets accèdent au canal de communication si la ressource est libre et attendent sinon. Les paquets d'une même application peuvent emprunter des chemins différents pour atteindre leur destination, ce qui crée une variation aléatoire de la latence. Les paquets peuvent également être perdus si aucune ressource n'est disponible.

Pour atténuer l'impact de la latence aléatoire et de la perte de paquets, les applications best-effort utilisent des mécanismes de type "poignée de main" (handshake) pour retransmettre les paquets perdus, ainsi que des "files d'attente tampons" (buffer) pour compenser la latence aléatoire. D'autre part, les fournisseurs de services Internet surdimensionnent souvent le réseau pour réduire la probabilité d'indisponibilité des ressources.

Afin de réduire les coûts et d'améliorer l'utilisation des ressources, le réseau IT s'est développé au fil des ans vers une architecture centralisée. Cependant, cette approche a introduit une latence inévitable due au temps de propagation, rendant l'implémentation d'applications temps-critiques impossible. Pour répondre à la demande croissante de faible latence, on observe aujourd'hui un développement opposé, avec la création de Data Centers plus petits, plus proches des utilisateurs finaux. L'utilisation des Edge Clouds permet également d'optimiser la bande passante en évitant de transmettre les données des utilisateurs jusqu'au Cloud central.

Avec l'accent désormais mis sur l'exécution d'applications à faible latence sur les réseaux IT, plusieurs groupes de travail ont été établis pour proposer de nouvelles normes permettant de transporter du trafic temps-critique sur l'infrastructure IT. Les groupes TSN (Time-Sensitive Networking) et DetNet collaborent pour créer des normes pour les couches 2 et 3 de l'architecture réseau, offrant une infrastructure fiable sur laquelle les applications peuvent être construites.

Avec l'amélioration de la vitesse et de l'efficacité de la commutation optique, les opérations dans des environnements dynamiques deviennent de plus en plus réalisables. Les chercheurs explorent la possibilité de redéfinir l'architecture des Edge Clouds en intégrant des commutateurs optiques plutôt que des commutateurs

électriques. La commutation optique rapide permet de partager les longueurs d'onde entre les points finaux du réseau, ce qui entraîne une utilisation plus efficace des ressources optiques. En tirant parti d'un contrôle déterministe de l'accès au canal, M. A. Mestre et al. proposent une nouvelle architecture pour les Edge-Clouds. B. Pan et al. ainsi que W. Lautenschlaeger et al. proposent des architectures pour la communication entre les Edge-Cloud.

La solution proposée pour fournir une très faible latence nécessite souvent une refonte importante de l'architecture réseau existante avec l'introduction d'équipements dédiés. Cependant, la majorité du trafic reste best-effort et passera par les Edge-Clouds. Le coût d'une nouvelle architecture et de son interopérabilité avec les réseaux existants pourrait être trop élevé pour ne traiter que 10% du trafic.

Aperçu du manuscrit et contributions

Dans le contexte d'un réseau convergé, intégrant les réseaux de télécommunications et OT dans les réseaux IT, l'objectif de ce manuscrit est de proposer une solution combinant la fiabilité développée au cours de nombreuses années dans les réseaux de télécommunications et OT, tout en restant aussi flexible et dynamique que les réseaux IT. Nous proposons une solution pour garantir une latence faible et stable à des clients sélectionnés, sur des réseaux existants ou des réseaux améliorés avec des technologies TSN.

D'autres objectifs incluent la surveillance de la latence et de la gigue dans des réseaux critiques pour vérifier qu'une application a respecté ses exigences. Nous fournissons une démonstration expérimentale de nos propositions avec une implémentation FPGA dans des expériences de conditions réseau, utilisant des équipements disponibles sur le marché.

Ce manuscrit est organisé en quatre chapitres, dans lesquels je présente le travail accompli pendant ces trois années de recherche.

Dans le premier chapitre, nous abordons les évolutions de la topologie du réseau. Nous passons en revue et définissons les exigences des applications temps-critiques. Ensuite, nous étudions les techniques existantes pour répondre aux besoins des applications temps-critiques. Enfin, nous définissons les performances visées dans ce manuscrit et le segment de réseau où nous faisons une proposition pour garantir une latence faible et stable.

Dans le deuxième chapitre, nous introduisons le système Determinist Dynamic X (DDX), un réseau distribué de systèmes conçu pour fournir une variation minimale de la latence dans les réseaux existants et actuels. Notre solution vise à satisfaire les

applications temps-critiques sans nécessiter une refonte complète de l'infrastructure réseau existante. Nous discutons de la distribution des cartes et du protocole que nous avons développé pour garantir une latence faible et stable. De plus, nous présentons notre mise en œuvre du protocole et démontrons ses performances à l'aide d'équipements disponible dans le commerce.

Dans le troisième chapitre, nous présentons notre solution de surveillance en-ligne et de haute précision que nous avons développée pour tester notre prototype. Nous introduisons un protocole de synchronisation pour fournir une haute précision et une tolérance aux pertes de communication. Nous démontrons les performances de notre système de surveillance mis en œuvre sur FPGA.

Enfin, dans le quatrième chapitre, nous approfondissons notre méthode pour fournir une latence stable et examinons le compromis obtenu entre la dynamique de la méthode et la précision de la variation de latence. Nous mettons en œuvre notre nouvelle méthode de limitation de la gigue réseau sur notre banc expérimental et mesurons le temps de mis-en-route du canal de communication déterministe ainsi que la précision de la variation de latence.

Contents

Acronyms

ATM Asynchronous Transfer Mode

CBOSS Cloud Burst Optical Slot Switching

CPRI Common Public Radio Interface

DCO Digitally Controlled Oscillator

DWDM Dense Wavelength Division Multiplexing

FPGA Field Programmable Gate Array

GA Growing Average

GPS Global Positioning System

HLS High Level Synthesis

IIR Infinite Impulse Response

INT In-band Network Telemetry

IT Information Technology

LLDP Link Layer Discovery Protocol

MA Moving Average

NTP Network Time Protocol

OAM Operations Administration and Maintenance

OT Operation Technology

OTN Optical Transport Network

P4 Programming Protocol-independent Packet Processors

PDF Probability Density Function

PDV Packet Delay Variation

PLL Phase Lock Loop

PON Passive Optical Networking

PSS Photonic Service Switch

PTP Precision Time Protocol

QoS Quality of Service

RAN Radio Access Network

RMS Root Mean Square

ROADM Reconfigurable Optical Add Drop Multiplexer

SDN Software Defined Networking

SyncE Synchronous Ethernet

TCP Transmission Control Protocol

TDM Time Division Multiplexing

ToR Top of Rack

TSN Time Sensitive Networking

VLAN Virtual Local Area Network

WDM Wavelength Division Multiplexing

WSS Wavelength Selection Switch

List of Figures

List of Tables

Introduction

The COVID crisis has highlighted how paramount the digital infrastructure has become not just to the global economy, but also to essential social interactions and well-being. The primary mission of this digital infrastructure is to carry data between two end points over a path, most of the time using light. For economic reasons, not each pair of end points owns its dedicated path for exchanging its data flow. Conversely, it shares a large fraction of this path with other flows. More specifically, data from different sources/destinations are aggregated (multiplexed passively and/or switched dynamically) into larger data streams and travel altogether through nodes. They are disaggregated into other nodes further down until they reach their destinations. The terminals, the nodes, the graph of paths and the intelligence to reconfigure it, is generally what is referred to as the "communication network".

Even though this communication network is seemingly one global network, it is more accurate to describe it as a concatenation of different networks, each optimized for performance and cost according to different sets of requirements. In particular, three network segments have been developing over time, almost in silos, driven by specific requirements, corresponding to specific market expectations. Naturally, the three segments have been built upon equally specific network technologies.

1. The Operation Technology (OT) network segment was developed for the closed environments of factories, for connecting, managing and monitoring different machines and sensors found on factory floors. These networks were built to ensure safety and reliability, guaranteeing a worker's safety and minimum downtime. These requirements and the slow evolution of needs in the factory floor made these networks rigid.

2. The Information Technology (IT) network segment has been designed to support data exchanges between computers and servers and has been optimized for maximum flexibility, as best response to the typical pattern of computer applications where end-points vary over very short lifetimes (e.g. often sub ms).

These networks are capable of connecting endpoints across the globe. Most commonly used in the IT networks, the Ethernet protocol and Internet protocol have been devised to enable packet routing, creating a dynamic network where endpoints can change location. These protocols however used a best-effort approach to access communication channels, limiting its reliability.

3. The Telecom network segment was first developed to carry messages, then voice communications over long distances. The voice communication over these networks introduced the need of a full path reservation to allow a proper reconstruction of the complete vocal communication. These requirements created a network that was not flexible with a scaling difficulty as the data rate increased. From these networks, we inherited the wireless communication networks to connect our phones to the internet. Going through different phases with different generations, until we arrive to the 5G with the wish to merge the Telecom network with the IT networks.

Among those three network segments, the IT network has seen the highest traffic increase [1] with bandwidth reaching 1,217 Tbps in 2023, mostly dominated by audio and video traffic [1], but also accommodating more secure applications like online banking. As Telecom networks, with the introduction of 5G, are becoming more flexible [2], and data processing and collecting in factories is being offloaded outside its premises [3], the IT network infrastructure is deemed to be the adequate support for those applications.

However, the OT networks and Telecom networks have constraints that the IT network was not built to meet. There are three main constraints which the applications, which we call time critical, running on these network require.

1. *Latency*: Applications may require information to be carried between different endpoints in a limited amount of time.

2. *Reliability*: Applications may tolerate a predefined amount of information loss throughout the communication.

3. *Jitter*: Applications may require the latency to be constant or to vary within a predefined interval.

When building the IT network, the focus was placed on dynamicity and flexibility. The network was devised to accommodate to short traffic flows (~1 ms). The access to the communication channel was performed in a best-effort fashion. In best-effort communication, packets access a communication channel if the resource is

free. Packets from a same application may take different path to their destination, creating a random latency variation. Packets can also be dropped if no resource is available. We call best-effort traffic, traffic which is not bothered by these drawbacks.

To mitigate the impact of random latency and packet loss, best-effort applications use handshakes to retransmit lost packets, and buffering to palliate random latency. On the other side, the internet service providers often over-dimension the network to reduce the probability of resource unavailability [4, 5].

In order to reduce costs, improve resource usage, the IT network developed for years towards a centralized architecture. However, this approach introduced unavoidable latency due to the propagation time which renders the implementation of time-critical applications impossible. To address the growing demand in low-latency, we nowadays observe an opposite development, where smaller data centers are created closer to the end-user. The use of Edge Clouds also helps in optimizing the bandwidth by not having to convey the user data up to the core cloud.

With the focus now on running low-latency applications on the IT network, multiple working groups have been established to propose new standards for carrying time-critical traffic on the IT infrastructure. TSN group and DetNet collaborate to create norms for layers 2 and 3 of the networking architecture, providing a reliable infrastructure upon which applications can be built upon.

As the speed and efficiency of optical switching is improving, operations in dynamic environments are becoming increasingly feasible. Researchers are exploring the possibility of redefining edge-cloud architecture to incorporate optical switches instead of electrical switches. Fast optical switching enables the sharing of wavelengths among network endpoints, resulting in more efficient utilization of optical resources. By leveraging a deterministic control of access to the channel, M. A. Mestre et al. [6] propose a novel architecture for edge clouds. B. Pan et al. [7] and W. Lautenschlaeger et al. [8] propose architecture for inter-edge-cloud communication.

The proposed solution for providing ultra low latency often require a significant overhaul of the existing network architecture with the introduction of dedicated equipment. However, the bulk of the traffic remains best-effort [1] and will pass through the edge-cloud. The cost of a new architecture and insuring its interoperability with existing networks might be too steep to tend to only 10% of the traffic.

Outline and contributions

In this context of a converged network, integrating OT and telecommunications networks into IT networks, the goal of this manuscript is to propose a solution

combining the reliability developed over many years of OT and telecommunications networks while remaining as flexible and dynamic as IT networks. We propose a solution to guarantee low and stable latency to selected clients over legacy networks or networks enhanced with TSN technologies. Other goals include the monitoring of latency and jitter in hot-network to verify that an application has met its requirements. We provide an experimental demonstration to our propositions with an FPGA implementation in network condition experiments with market available equipment.

I conducted my PhD research under an industrial grant, collaborating with Nokia Bell Labs (Nozay, France) and the Information Processing and Communications Laboratory (LTCI) of Telecom Paris (Palaiseau, France). This manuscript is organized into four chapters, in which I present the work accomplished during these three years of research.

In the first chapter, we discuss the evolutions in the network topology. We review and define the requirements of time-critical applications. Then we will study the existing techniques for catering to time-critical applications. Lastly we define the targeted applications for this manuscript and the network segment where we will make a proposition for guaranteeing low and stable latency.

In the second chapter, we introduce Determinist Dynamic X, a distributed network of systems designed to provide minimal latency variation in legacy and current networks. Our solution aims to satisfy time-critical applications without requiring a complete overhaul of the existing network infrastructure. We will discuss the distribution of the cards and the protocol we have developed to guarantee low and stable latency. Additionally, we will detail our implementation of the protocol and demonstrate its performance using commercial equipment.

In the third chapter, we present our in-band and high-precision monitoring solution we developed for testing our prototype. We introduce a synchronization protocol to provide high precision and link failure tolerance. We demonstrate the performance of our monitoring system implemented on FPGA.

Finally, in chapter 4, we delve deeper into our method for providing stable latency, and examine the trade-off obtained between dynamicity of the method and the precision of the latency variation. We implement our new method for limiting network jitter on our FPGA test set-up and measure the rise time of the deterministic communication channel and the precision of the latency variation.

Publications

1. Nihel Benzaoui, **Guillaume Soudais**, Olivier Angot, Phillipe Robineau, Sebastien Bigo, "DDX add-on card: transforming any optical legacy network into a deterministic infrastructure," in 2021 European Conference on Optical Communication (ECOC), Bordeaux, France: IEEE, Sep. 13, 2021, pp. 1–4, ISBN: 978-1-66543-868-1. DOI: 10.1109/ECOC52684.2021.9605880

2. **Guillaume Soudais**, Sébastien Bigo, Nihel Benzaoui, "Per packet distributed monitoring plane with nanoseconds measurements precision," in 2021 European Conference on Optical Communications (ECOC), Bordeaux, France: IEEE, Sep. 13, 2021, pp. 1–4, ISBN: 978-1-6654-3868-1, DOI: 10.1109/ECOC52684.2021.9605806

3. **Guillaume Soudais**, Tarik Graba, Yves Mathieu, Sebastien Bigo, "Jitter compensation mechanism for dynamic deterministic networks," in Optical Fiber Communication Conference (OFC) 2023, San Diego California: Optica Publishing Group, 2023, Th3D.2, ISBN: 978-1-957171-18-0, DOI: 10.1364/OFC.2023.Th3D.2

Patent

1. Nihel Benzaoui, Sebastien Bigo, **Guillaume Soudais**, "Deterministic Communications via Packet-Switched Networks", United States Patent and Trademark Office, 28th Feb 2022.

Chapter 1

Performance guarantee in a converged network

Contents

1.1 Introduction

A convergence of network is now observed as the development of 5G technology and the processing of factory data inside edge-clouds have led to applications running on telecommunications and OT networks to be deployed to a segment traditionally dedicated to IT networks. The IT network has high resources however it is unable to provide quality of service to the applications depend on it. Applications originating

from the OT and telecommunications networks have stringent requirements that fail to be met by the current IT networks.

In this chapter, we will define the metrics crucial for time-critical applications and review example applications along with their requirements. We will then outline the network architecture upon which these applications are envisioned to operate. Following this, we will present existing projects and techniques aimed at providing network guarantees for traffic. Finally, we will specify the targeted applications and network that we aim to address within this manuscript.

1.2 Performance guarantee for time-critical applications

Due to the unique and demanding requirements of OT and telecommunications network applications, specialized networks have been developed to cater to their specific needs. These applications often involve time-critical processes, where even milliseconds of delay can have significant consequences for system performance, reliability, and safety. To ensure the efficient and reliable operation of these applications, it is crucial to have well-defined performance metrics that can guide the evaluation, comparison, and optimization of various network solutions.

In this section, we will define and discuss several key metrics that are particularly relevant to time-critical applications in OT and telecommunications networks. Then a review of example applications will give us the target metrics for the development of our proposition.

1.2.1 Performance metrics

In the industry, when building a product, a machine is programmed to perform a certain set of tasks. In order to increase the speed of production, the revolution proposed by the Industry 4.0 is for all the machines on the factory floor to intricately cooperate. To achieve this cooperation, machines must share their progress in completing a task and be able to remotely receive orders.

In order to exchange information, machines are connected on a network. The network allows for a communication between machine and also with servers where the computation is performed to give a new set of orders. We define network more generally outside the scope of Industry 4.0 as the collection of interconnected devices, systems, or components that are linked together to facilitate communication. In this

manuscript we often refer to the device connected on networks as server because many time-critical applications require computation performed on servers.

On the servers, the tasks running to perform computation or give orders to machines are called applications. In the studied networks these applications exchange information via packets, a small bunch of data. Applications send their packet on the network to be routed to their destination. It is in the network, where packets of many applications in parallel travel, that packets can be lost or delayed due to the contention created by the traffic overload.

In this manuscript, we want to provide performance guarantee for applications at a network level, meaning providing guarantees in the transfer of their packets.

In the convergence of network and the evolution of applications we can divide the requirements into two distinct categories, dynamic and deterministic. Deterministic requirements originate from the OT and telecommunications applications' legacy. The dynamic aspect, on the other hand, arises from the new applications and the objective in merging the different network segments in the IT network.

There are three main characteristics to a deterministic network, coinciding with the requirements of time critical applications highlighted in the Introduction.

Reliability is a key metric at the network level for application, it is described by the packet loss ratio of the network. Virtually zero packet loss is tolerated for time-critical applications ($< 10^{-10}$ loss ratio). Typically, in IT networks the packet loss ratio is a lot higher, around 10^{-3}. To mitigate the potential packet loss, applications can use protocols like Transmission Control Protocol (TCP), in order to retransmit packets if no acknowledgment has been received from the destination within a lapse of time. This help reducing packet loss but with a great impact on latency and jitter.

Time-critical applications need their packets not to be lost, and the frontier between a packet with an abnormally long latency and a packet loss is thin. Hence, time-critical applications require their packet to be transferred on the network with a bounded latency. Often, the sooner the packet arrives the better it is for the application, so the lower bound of the latency can be zero second. Then, it is up to the application to define an upper-bounded latency for the proper execution of the process. We give example of this metric in Table 1.1.

Other applications, often for synchronization, may require a minimal latency variation, the difference between the lower and upper bound latency to be small. We call jitter the variation of latency. In this manuscript we will consider RMS jitter, the standard deviation of latency over a given period of time. We will also use peak-to-peak jitter as the difference between the maximum and minimum of

latency over a given period of time. Peak-to-peak jitter helps to take into account the abnormal events during transmission.

The norm RFC3393 [9] defines Packet Delay Variation (PDV) as the latency difference between two given packets. Often jitter is described as the standard deviation of the latency distribution. This is the definition used by our measurement system in the lab. However, this definition fails to account for sporadic events if there are too infrequent. Therefore, in Chapter 4, we will redefine jitter as the maximum PDV over a given period of time.

As deterministic applications transition to a new segment with a dispersed distribution of edge clouds, applications are likely to move between locations. The emergence of shorter applications, with milliseconds lifetime, with deterministic performance requirements also highlights the need for dynamicity. To prevent the network from becoming a bottleneck, it should be dynamic, meaning that the turn up/tear down of processes and the reconfiguration speed of the network should happen in milliseconds or lower.

We can define the turn-up time as the duration required by the network between receiving a request from an application and being able to carry said application while respecting its timing requirements.

The tear-down process is the counterpart to the turn-up time, referring to how quickly the network can release the resources of a completed application. Additionally, the network should be able to adjust its resource allocations quickly, ideally faster than it takes to initially allocate them.

1.3 Time sensitive applications

Time-sensitive applications come from various backgrounds. Mainly as the transfer of applications that used to run in the OT and telecommunications networks. These applications are now set to run in the edge cloud. In this section we will present, separated in two groups, industrial time sensitive applications and computing in the edge cloud, followed by applications related to the deployment of 5G.

1.3.1 Edge-cloud applications

The evolution of production methods, such as the generalization of programmable hardware controlled by dedicated software [10], has enabled software-defined manufacturing. This development has enabled industrial applications to be processed

remotely, in the edge cloud, thereby increasing the need for time-sensitive networks close to the end-users.

We compile in Table 1.1 the requirements of time-sensitive applications in terms of latency, jitter, data rate and reliability.

Applications	Latency	Jitter	Data Rate	Reliability
Electricity distribution - medium voltage[11]	40 ms	NA	10 Mb/s	99.9%
Electricity distribution - high voltage [11]	5 ms	NA	10 Mb/s	99.999 %
Interconnections of extra high voltage electric substations for WAN deployment [12]	5 ms	10 µs	NA	99.99999 %
Current differential protection [12]	5 ms	250 µs	64 kb/s	99.9999 %
Periodic smart factory control to control[11]	4 ms	2 ms	1 kb/s	99.99999 %
Machine-to-machine time synchronization [12]	1 ms	1 µs	NA	99.999 %
Automated guided vehicle [11, 13]	1 ms	$<\frac{1}{2}$ latency	10 Mb/s	99.9999 %
Enhanced distributed computing[14]	10 ms	<1 µs		
Remote surgery [15, 16]	1 ms	500 µs	150 Mb/s	99.9999 %

Table 1.1: Performance requirements for low latency and high reliability applications.

We can see that most of the time sensitive applications are control loops. They expect delivery and time guarantees to provided safety to equipments and people.

In most applications, the maximum allowed latency is in the order of milliseconds. The tolerable jitter for these applications can range from microseconds to a millisecond, depending on the specific activity. For instance, in the case of remote surgery, one might assume that network uncertainty, jitter, should be minimal due to the lives at stakes. However, the allowed jitter is of the same order of magnitude as the latency. Since a surgeon's reaction time is one to three orders of magnitude slower than the allowed jitter, this difference is unlikely to be noticeable.

We observe the same phenomenon for control loops of machines, because they as well possess an intrinsic reaction time. The jitter constraint is increased for the communication between services. Lower jitter increases the precision of synchro-

nization in networks, for industrial machines this can translate in tighter operation which will improve the completion time of a task. The same principle applies to edge data-centers, where lower jitter through predefined access to network resources can achieve a 70% time gain in the completion of distributed computation processes [14].

These applications share a common characteristic: they have a very low tolerance for packet loss. In a converged network, we need to combine the flexibility of IT networks with the reliability of OT and telecommunications networks. Through these examples, we have explored the requirements for applications coming from the industry evolution and edge-cloud deployment, we saw the needs of applications previously running on the OT networks. The telecommunications networks also want to leverage the huge resource potential of the IT networks.

1.3.2 Cloud Radio Access Network

The telecommunications networks are evolving with the goal to bring an increased bandwidth to the user and to accept the ever-growing number of connected object. To be able to increase the bandwidth, a new protocol and a new frequency range for the wireless communication is being used. However, the increased bandwidth comes with a drawback of a reduced range, the propagation of higher frequencies through the air does not reach as far as the previous technologies. A 4G antenna could previously cover distances of 2 to 40 km, depending on terrain and population density, when the new millimeter wavelength used in the 5G communications only covers a limited range, one hundred meters to a kilometer depending on the frequency, due to the air opacity at these frequencies and the low penetrability in buildings.

In previous wireless communications, the antenna is colocated with a server, called base station, which converts the radio frequency signal into Ethernet packets to be sent on the Internet. The communication between the antenna and the base station is performed with the Common Public Radio Interface (CPRI) protocol [17].

To compensate for this reduced reach more antennas need to be placed to cover the same area. Having more antennas in the same area also means that the user will switch antennas more often as it moves through the landscape. Thus, for the 5G deployment, the network architecture was rethought to leverage these new issues. A proposed development is to gather the base stations of all the antennas of an area into a single facility. This architecture enables resource sharing between antennas and facilitates the transfer of users between antennas [18].

The communication link between antenna and base station which was previously only a few meters and a point to point connection suddenly becomes a link of multiple kilometers potentially shared between multiple antennas. The standard for radio

communication has evolved into eCPRI [19], a packeted version of CPRI to improve the resource utilization of the link between antenna and base station.

The new eCPRI protocol has shifted the frequency usage from Wavelength Division Multiplexing (WDM) in previous wireless communication to Time Division Multiplexing (TDM) in the new 5G protocol. This change enhances the resource utilization of the entire frequency spectrum but requires the antennas to be synchronized to avoid simultaneous transmissions, which could cause interference.

Category	TAE
A+	65 ns
A	130 ns
B	260 ns
C	3 µs

Table 1.2: Class of services for 5G eCPRI links. (TAE : Timing Alignment error)

In the definition of the eCPRI protocol [19], multiple categories are defined based on the level of synchronization achievable between the antenna and the base station. In Table 1.2, we list the different categories showing the requirements of 5G. In its most degraded category, eCPRI still have the same requirements as the most stringent time critical applications hosted in the edge cloud detailed in Table 1.1, allowing for a maximum timing alignment error of 3 µs. Since the synchronization is performed over the Ethernet link between the antenna and the base station, the jitter on this link is responsible for the degradation of the synchronization.

The convergence of network segments has brought on the IT networks the requirements of time-sensitive applications originating from the telecommunications and OT networks. The offloading of industrial computation requires jitter close to the microsecond, from the factory floor to the servers inside the edge cloud, and requires an extremely low packet loss. Simulations have also shown that reduced jitter in the edge cloud could enhance the efficiency of shared computing[14]. The most stringent requirements stem from the evolution of telecommunications networks, regrouping base stations in edge clouds.

In this manuscript we aim to propose a solution to provide reliable communication for time sensitive applications, providing jitter below 100 ns. This will facilitate the integration of time-sensitive applications coming from the OT network in the IT

network. Additionally, our solution will be capable of providing the second-best class of service in the eCPRI protocol.

1.4 Bringing the time critical applications to the edge cloud

Reducing latency has always been a priority in developing network, be it for providing a better experience to the user. However, the evolution proposed by the convergence of network is demanding unprecedented networks improvements.

In the past, technologies have tried to provide low latency with minimal jitter, Asynchronous Transfer Mode (ATM) was developed to leverage the IT network to provide controlled latency, reaching few microseconds of packet delay variation [20]. ATM was quite successful for performance and efficiency but failed on the dynamics of connectivity, where Ethernet shined the most.

Nowadays, the development of dynamic and determinist networks is driven by several factors, including the convergence of networks, the industry's desire to offload computation in the edge cloud, the deployment of the 5G technology with remote signal processing, and the improvement of distributed computing by treating the network as a resource. These advancements necessitate a reevaluation of our network protocols.

1.4.1 Time sensitive networking

There has been a global effort to provide solutions for network communications with lower latencies and jitter. The Time Sensitive Networking (TSN) group has proposed sets of norms aimed at enhancing the Ethernet protocol. The focus of the TSN task force revolves around three main pillars:

- Time Synchronization
- Scheduling and Traffic Shaping
- Selection of Communication Paths and Fault Tolerance

The Time Synchronization for TSN (IEEE 802.1AS [21]) standard defines a protocol for synchronizing clocks. The protocol extends the Precision Time Protocol (PTP)[22], which is a widely used standard for synchronizing clocks in distributed networks. To ensure precise synchronization, IEEE 802.1AS supports a number of features, such as path delay measurement, clock quality monitoring, and support

for multiple time domains. These features enable TSN networks to achieve sub-microsecond synchronization accuracy, even in complex network topologies.

The goal of scheduling and traffic shaping is to offer the possibility for different types of traffic with different type of priority to coexist in the same network. There are various norms proposed in TSN to achieve network scheduling, among which is norm IEEE 802.1Qbv [23], which describes a time-slot division with repeated periods for the end-to-end communication path. Time slots can be attributed to different classes of traffic, enabling conflict free transmission between time-critical and best-effort traffic. However, there is a commonly seen issue with scheduled network akin to coin-base access networks is the poor use of resources to the detriment here of best-effort traffic. Depending on the TSN implementation, synchronous or asynchronous, the effect on best-effort traffic can be very different [24], the quality of service deteriorates in different ways with the TSN implementation and the traffic shape. This study highlights the need to research the tuning of TSN parameters for the different use cases.

Another notable proposition from TSN to shape traffic is norm IEEE 802.1Qbu which offers the possibility to interrupt the transmission of best-effort packets to send time-critical packets and resume the transmission afterward. In this manuscript we are particularly interested in this norm and will propose a solution to build deterministic networks using it.

The third pillar offers reliability through the selection of a communication path and fault tolerance. The norm IEEE 802.1CB provides dual-link redundancy over Ethernet, with packet duplication at the source and elimination at destination, ensuring that a single packet is received.

Though many deterministic protocols are at the development stage, we can find implementations of some norms created by the TSN group in commercialized products. We see features of preemption detailed in the norm IEEE 802.1Qbu in the 1830 time-sensitive packet switch from Nokia.

1.4.2 Time sensitive data centers

Alongside TSN, propositions have been made to build networks reducing latency variation under few hundreds of nanoseconds. Cloud Burst Optical Slot Switching (CBOSS) is the proposition for an edge cloud architecture. To provide deterministic performances to various time critical applications, nodes are placed on a round path in order to guarantee reliability with possible redundancy in both direction of the circle. Each node serves as a Top of Rack (ToR) switch in a usual data center architecture to the rack of connected servers.

CBOSS leverages optical components and Software Defined Networking (SDN) network architecture and scheduling to provide deterministic latency for concurrent time-critical applications. Through the use of fast-tunable lasers (4 ns) and Wavelength Selection Switch (WSS) they are able to share wavelength between nodes limiting the loss of resources due to the scheduling of the communication path. In a prototype they achieve sub-100 nanoseconds of latency variation on a synchronized ring.

For bigger data centers, a toroidal architecture is envisioned as the imbrication of multiple rings.

1.4.3 Interconnecting time sensitive data centers

While some time-sensitive applications may require guaranteed performance within the confines of a single data center (e.g. synchronized shared computing [14]), most other applications shown in table 1.1 require performance guarantees spanning multiple network location. It can be the case for Industry 4.0 application, where end-to-end performance guarantees are required between the factory floor and edge clouds, or the distributed network of antennas for 5G communication.

We therefore need the IT network to provide service guarantee on a larger scale, across data centers in the edge network.

In some situations, Passive Optical Networking (PON) can be a solution to provide service guarantee for multiple path. Through the use of Dense Wavelength Division Multiplexing (DWDM), many point-to-point high bandwidth channel can be created. On these channels the absence of contention allows providing latency variation around few tens of nanoseconds. However, Levrau et al. studied [25] cost of different technologies and passive technologies were most cost-effective for low density connections. Alternate technologies presented better resource utilization when then number of cell site or antenna increased.

In order to share optical resources, Optical Transport Network (OTN) was developed. OTN uses Time Division Multiplexing to carry independently traffic from multiple sources on a common optical path. While for established route the latency variation can be kept below 20 µs and can be compensated to reach a lower bar, OTN lacks the dynamic side of a technology we aim for, with turn-up times around a second.

To achieve determinism and dynamicity, rapid wavelength switching in optical rings is a promising architecture. B. Pan et al. demonstrated[7] below 12 ns time accuracy for a ring of four nodes and distances on a city scale. To limit the scalability issues in scheduled network, Lautenschlaeger et al. are proposing to limit the size of

TSN networks [26]. Creating TSN islands interconnected a deterministic bus called Industrial Optical Ethernet.

1.5 Conclusion

The proposed solutions advocate for time-synchronization across a whole network which can become a scalability issue when the number of nodes increase, especially in rings where synchronization is performed neighbor to neighbor. We will present a solution which does not require neighbor to neighbor synchronization whilst still being able to provide low latency variation.

Moreover, they propose a new network architecture, where network equipment needs to be replaced to cater to time critical clients. We think that a network overhaul represent too great of a cost for time-critical applications that represent only a fraction of the traffic in the network close to the user.

In this work we aim to provide guarantee with the minimum impact on the rest of the traffic. We will present a solution to provide latency guarantee for legacy networks, only adding equipment where deterministic performance are needed.

Chapter 2

Determinist Dynamic Network node (DDX): Protocol and Implementation

Contents

In Chapter 1, we discussed how diverse time-critical applications possess specific requirements that are often unmet by current network infrastructures. We explored various solutions introduced to address the demand for both deterministic and dynamic performance. However, many of these solutions propose the implementation of an entirely new network architecture.

The vast majority primarily transport and process video data [27]. They allow for buffering over multiple seconds at the endpoint. Designing an architecture capable of accommodating multiple traffic classes, including video services is crucial for effectively deploying future technologies. We believe that completely altering the network architecture and hardware to cater to time-critical applications, which account for only a small fraction of the total network traffic, might not be justifiable.

In this chapter, we propose a solution to enhance packet networks by enabling the transport of time-critical applications. We develop our solution to be compatible with legacy network and improved by the current network protocol evolution like TSN.

We develop a protocol to ensure a determinist latency through a network. Because legacy equipment do not offer the possibility to have such capability, we develop this protocol over specific hardware extensions with the goal of being modular and pay-as-you-grow. We install our equipment only where needed. In future networks, our protocol could be implemented inside switches or network cards in servers.

2.1 DDX node network positioning

The protocol that we will detail afterwards enables the creation of a deterministic path between two endpoints, with the possibility to add intermediate nodes on the path. Our approach is to build on the current network by enhancing its capacity to accommodate for time-critical traffic. We choose a modular approach by implementing our protocol in add-on cards, DDX add-on cards, which can be placed anywhere in the network. In order to decide where to place our add-on cards, we first need to identify the path over which we are going to provide a controlled latency.

With the development of Industry 4.0, more computing power is expected to be moved into the edge cloud and needs a deterministic path between the factory floor and the dedicated edge data center.

For the variety of network technologies that time-critical applications can cross, we aim to be technology-agnostic. The network to bridge can use transport technologies as OTN for inter edge-data center crossing, Reconfigurable Optical Add Drop Multiplexer (ROADM) for inside or outside of data centers, or just plain Ethernet network.

We can identify the link over which time-critical traffic will flow. Once the path has been identified, we place our equipment to ensure a controlled latency over this predefined path.

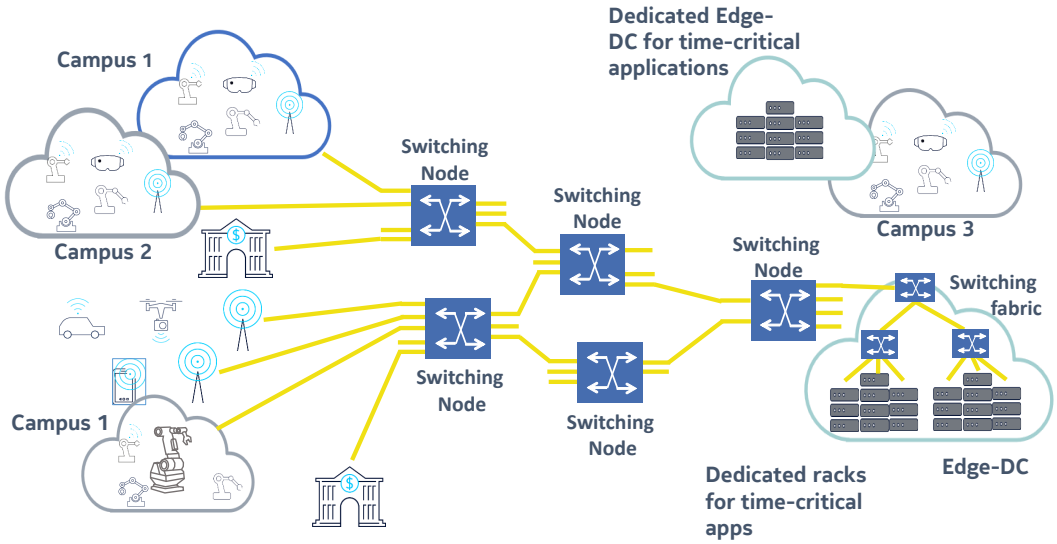

Figure 2.1: Network architecture for carrying time-critical applications

Figure 2.1 depicts such a network configuration where the industrial resources placed in Campus 1 and Campus 2 need a deterministic link to some computing resources in an edge cloud.

In order to provide low and controlled latency, we insert the DDX add-on card on the first and last node of a communication path. We locate our DDX card as close as possible to the targeted servers, ideally between the server and the top-of-rack switch, so that no other equipment can add jitter which cannot be compensated.

This configuration is depicted in the upper part of Figure 2.2 where the DDX add-on cards will create a deterministic path between the industrial campuses and the dedicated edge resources over the existing network.

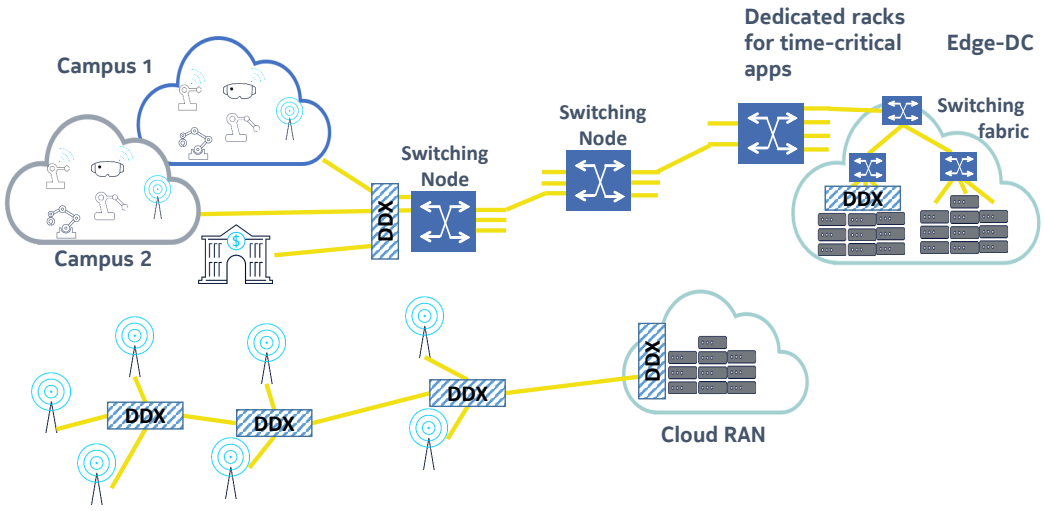

Figure 2.2: Insertion point of DDX add-on cards

We can place intermediate DDX nodes over the deterministic path to increase the number of insertion points. This scenario is depicted in the lower part of Figure 2.2 where multiple traffic insertion points are present along the deterministic path. In this scenario, we gather the traffic from multiple 5G antennas to route it toward a cloud Radio Access Network (RAN) where are deployed the remote base stations.

2.2 Overview of the DDX protocol

In order to enhance networks, we developed the DDX protocol to primarily meet three key features for providing low and controlled latency. In Figure 2.3, we depict the layout of a typical network that we propose to manage time-critical applications through a deterministic bus transporting one or multiple time-critical clients, from an ingress node to an egress node.

First, the multiple time-critical clients are aggregated within the ingress DDX card to form a single flow of packets. We perform the aggregation according to a schedule computed by a network orchestrator. Preemptively, the time-critical applications send a request to the network orchestrator for an allocation of a predefined throughput over the deterministic bus.

Second, we perform network bridging in every switch or router over the path computed by the orchestrator. We call bridging a configuration which ensures data delivery up to a maximum and guaranteed latency between two ports of a switch or router. This enables us to provide deterministic performance for our clients. We call

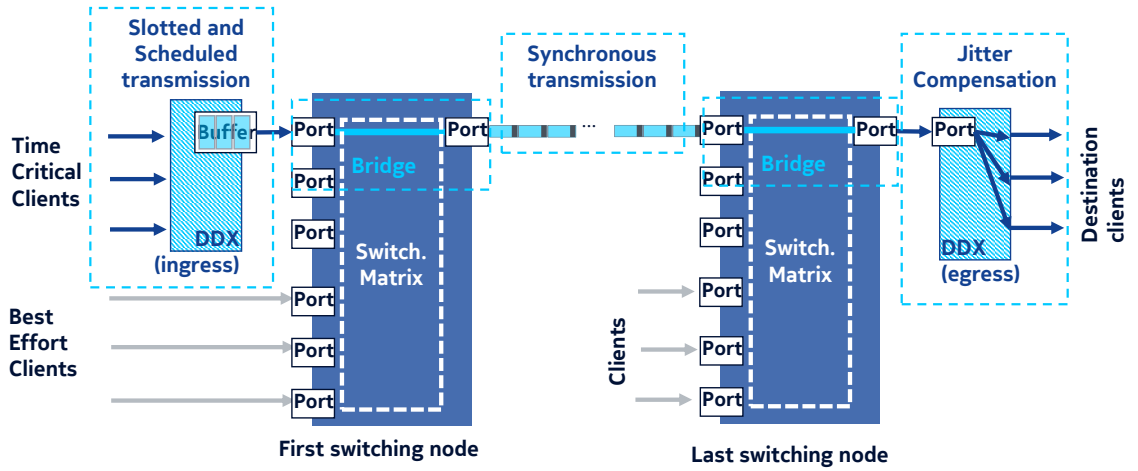

Figure 2.3: Three main features of the DDX add-on card: Insertion, transmission and Extraction

this path the deterministic path. On this path will flow the packets containing the data of time-critical clients.

Third, at the end of the deterministic path, in the egress DDX card, we perform jitter compensation. We equalize the latency of every packet of a time-critical client traffic flow by trading marginal latency.

We perform this operation inside our card, by slowing down packets which arrived earlier than the guaranteed upper bound latency. Guaranteeing the same latency for every packet enables the use of synchronization over our deterministic bus. Here, the charge of providing controlled latency is fully carried out by our equipment. We can imagine an alternative scenario where the application takes a part of this burden for its own benefit.

For example, instead of performing jitter compensation in the egress DDX, we could send the packet as soon as it arrives at the egress node with the timing information of when the packet would have arrived with jitter compensation. This method grants the minimal latency possible with a higher apparent jitter, which can be reduced at the software level.

In this manuscript, we focus on the first scenario, performing the jitter compensation inside the DDX card. The latency packet latency is increased by a few microseconds, but no new software is needed in the time-critical clients to ensure ultra-low jitter. It should be emphasized that all three key features have been devised to be dynamically reconfigurable, so that any deterministic path can be quickly allocated and released on demand. Hence, when a time sensitive application vanishes, resources are available again for native best-effort traffic (e.g. video).

2.3 Engineering a deterministic path through the network

The DDX concept relies on enhancing the performance of a predefined path through a network. As we aim at building this concept upon legacy networks with minor modifications to the existing architecture, we have to achieve such a path over existing products.

In today's networks, the main source of latency variation and uncertainty in data delivery comes primarily from network congestion. When multiple flows try to access the same port, one or several flows undergo a delay which adds to the end-to-end latency, creating latency variations over time. In the worst scenario, one flow can even be interrupted due to bandwidth insufficiencies creating packet loss. This effect adds up at every point of switching, and creates a tail in the distribution of latency with an unbounded arrival time.

Legacy networks are not equipped with any strict mechanism against congestion that would achieve a bounded latency. Therefore, we need to make sure that congestion never builds up. The goal is for all traffic emanating from our card not to encounter any competing traffic on the path we are building toward the network egress.

When receiving a request for a new path, the DDX node contacts the orchestrator to create a new path through the network. In a process similar to SDN, the orchestrator communicates to every node on the path to create the bridge ensuring that no competing traffic will have access to the port used by the DDX bus. A similar process is used to terminate the path and release the corresponding resources for best-effort traffic. Parts of the orchestrator may be implemented in the ingress node, with the job of configuring every switch on the deterministic path as highlighted in Figure 2.4.

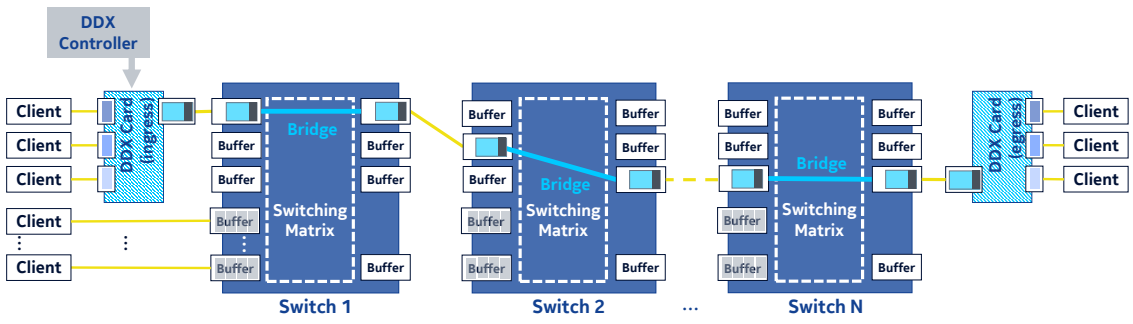

Figure 2.4: Network orchestrator reserves full path to ensure determinism

In order to bridge the path between the two DDX nodes we propose different methods, creating a Virtual Local Area Network (VLAN) over the whole path and having the DDX traffic be the only allowed traffic in this VLAN; using preemption as defined by the TSN taskforce in norm 802.1Qbu [28], and have DDX traffic be express.

For longer distance travel, other protocols can be used. We can leverage the Optical Transport Network (OTN) protocol which is a deterministic protocol providing guaranteed delivery.

2.3.1 Building a VLAN bridge

In order to maintain a strictly controlled quality of service over legacy networks, we have to exclude any competing traffic to travel over the full preselected path or any portion of this path. By contrast, the usual approach for improving Quality of Service (QoS) consist of discriminating between different classes of service. It leads to a tail distribution with a smaller standard deviation but cannot guarantee a maximum latency. Therefore, we eliminate competition by creating a VLAN. Only the packets tagged with the designated VLAN are authorized to enter the tagged ports. This is a complete reservation of the path, no other traffic than the DDX frames are allowed to travel on the bridge.

As there is no competition, the DDX frames is directly forwarded at the switch and the overall latency of the path turns into the sum of the forwarding times in the switches and the transit delay of transmission between every pair of switches along the path. The forwarding time of the switches is however subjected to some variations. We have measured a typical variation of 250 ns[1] when crossing commercial Ethernet switches of various types. As a result, the more switches on a deterministic path the more latency variation will be added.

Though latency variation is added, the variation remains bounded because there is no competition with other sources of traffic, hence the delivery time of packets can be guaranteed when jitter compensation is performed.

In order to provide deterministic latency, each bus will need its own VLAN. Having a complete exclusion of other traffic will naturally limit the performance of the network for other applications[2].

[1]Latency variation of one 1 Gbps CBR traffic flow through a 10 Gbps VLAN in a Nokia TPS 1830 switch.

[2]Networks are often built with an overdimensionning of resources probably limiting the impact of the reservation. However, further studies need to be done to evaluate the impact of reservation on best-effort traffic

2.3.2 Leveraging preemption to create a bridge

Creating a VLAN dedicated for the DDX traffic provides low but cumulative latency variation. However, it completely excludes best-effort traffic from a network path which creates a strain on the rest of the network. For the convergence of IT and OT, the Time Sensitive Networking (TSN) community has developed methods for isolating critical traffic from best-effort traffic. These methods have been standardized and some are available in commercial product.

In legacy Ethernet, access to the communication channel between different clients is computed a single time at the end of each transmission. The access is then granted following different algorithms often based around class of service and built to limit the possibility of client starvation. TSN brings forward preemption which allows for critical traffic to interrupt the transmission of a best-effort packet. Thus, it can be ensured that the critical traffic will always have access to the communication link and a maximum latency can be derived.

In the norm 802.1Qbu defining preemption, two classes of traffic are described. First, *preemptable traffic*, the class of service which can be interrupted, and second, *express traffic*, which can interrupt the first class of traffic. For this system to be efficient, there cannot be multiple uncoordinated sources of express traffic, else we encounter the same problems as legacy Ethernet network, multiple services competing for a single resource, creating unbounded latency. Therefore, all express traffic must be orchestrated in a single flow for preemption to be efficient. In TSN, the norm 802.1Qbv describes reservation and cyclic schedule gating providing an isolation of critical traffic. This condition is met with our DDX concept, where we perform scheduling and insertion of multiple time-critical clients into a single flow. We present our method in section 2.4.

Preemption adds more latency variation than a complete reservation of the communication link via a VLAN. At each hop, similarly to a VLAN reservation, the equipment adds latency variation, mostly due asynchronous clocks between two endpoints. With preemption comes a second source of added latency variation. Standards impose a protection mechanism whereby preemptable traffic cannot be interrupted during the transmission of the first few bytes, for a maximum non preemptable fragment of 64 bytes, and during the final 64 bytes of a packet. This protection is placed to ensure the complete transmission of The Ethernet header of the preemptable traffic, and the valid completion of a packet. The latency variation added by this mechanism varies based on the throughput, for 10 Gbs Ethernet, 128 bytes corresponds to a waiting period of 100 ns. Because the express frame can

arrive at any time during the non-preemptable time window, this mechanism adds latency variation, bringing the overall latency variation to 350 ns[3] per switch when preemption is activated.

This relatively small increase of latency variation is generally worth paying for as it comes with increased resource utilization that greatly benefits to best-effort traffic which is no longer excluded from the deterministic path. Note that the latency of best-effort traffic through the preemptable channel is slightly increased when compared to another channel without preemption.

However, we can create a class of service in the best-effort traffic that has an acceptable quality of service. With a maximum non-preemptable fragment of 64 bytes and the non-preemptable 64 bytes at the end of the packet, packets less than 128 bytes long can never be interrupted. From their point of view, the express channel acts as a higher class of service channel, and the latency of the packets from this class of service will not suffer a large increase. So, in this set up important but not critical information can be transported inside packet smaller than 128 bytes.

A word of 128 bytes represents a small payload. 802.1Qbu standards allow for customizing the maximum non-preemptable fragment of packet. 64 bytes being the minimum, ensuring the minimum latency variation added due to preemption. We can trade the added latency variation for quality of service for small packets. The maximum non-preemptable fragment size can be chosen between 64, 128, 192 or 256 bytes, thus allowing for longer packets to pass through unfragmented. The contribution of preemption in the latency variation added by hop can increase to 250 ns giving an overall latency variation of 500 ns per hop.

2.3.3 Leveraging network transport technologies

We have seen methods for creating a path with bounded latency through Ethernet networks, either by creating a VLAN or by having an express route dedicated for DDX traffic. As the telecommunication networks converges with IT and OT, data communication can start using telecommunication networks, the OTN protocol can serve as a deterministic bus linking multiple sites. Though OTN is slowly reconfigurable, it provides guaranteed resources through the reservation of containers on a time division multiplexed communication channel.

OTN provides low and bounded jitter. In an experiment we measured a jitter of 20 µs. This jitter is created by the uncompensated wait period in the ingress OTN device when waiting for the periodical reservation.

[3]Latency variation of one 1 Gbps CBR express traffic flow through a 10 Gbps network port with four 1 Gbps ON/OFF normal traffic for contention in a Nokia TPS 1830 switch.

2.3.4 Choosing how to build a bridge

In this section we described the types of bridge we could form to provide guaranteed delivery and a bounded maximum latency necessary for the jitter compensation we are going to perform afterwards. In the following sections we perform tests over those three types of bridges. We measure the performance that our implementation makes possible.

One might choose a bridge type for different reasons, and deterministic paths can even be built combining multiple technologies.

In most cases, in cloud applications, an Ethernet network will be deployed with legacy switch. Therefore, creating a VLAN for the time-critical traffic seems like a relatively straightforward approach. It brings the least amount of latency variation that needs to be corrected, but it takes over the whole communication link, limiting the possibility to reroute other best-effort traffic. Thus, it could affect the network scalability when the number of deterministic buses increases. This deterministic bridge can be quickly set up, in less than a millisecond, time for the orchestrator to send commands to configure the switch to create the VLAN service.

With new switching equipment entering the market, the implementation of the preemption feature of TSN is becoming more common. We believe that the use of preemption is preferable to a full deterministic bridge as the impact on the network will be minimal. We only need to send the time-critical traffic and timing information comprised in the DDX protocol.

For insuring a bounded latency between edge cloud data-centers we might want to leverage the telecommunications equipment that provide long reach with low latency variation.OTN is less flexible than Ethernet and is slowly reconfigurable but provides deterministic services in guaranteeing bandwidth to each client in its schedule.

2.4 Multiplexing of time-critical clients

The reservation of a deterministic path through a network gives us a path upon which to send the packets of time-critical clients. Where legacy equipment has the opportunity of providing bounded latency to a single client, we leverage this opportunity to share it among time-critical clients.

We perform time division multiplexing to integrate all the time-critical traffic flow into a single packet flow sent over the deterministic path. In Figure 2.5, we show the connection between the time-critical clients to the ingress DDX node. Packets from the time-critical clients are first stored in buffers until their reserved slot arrives.

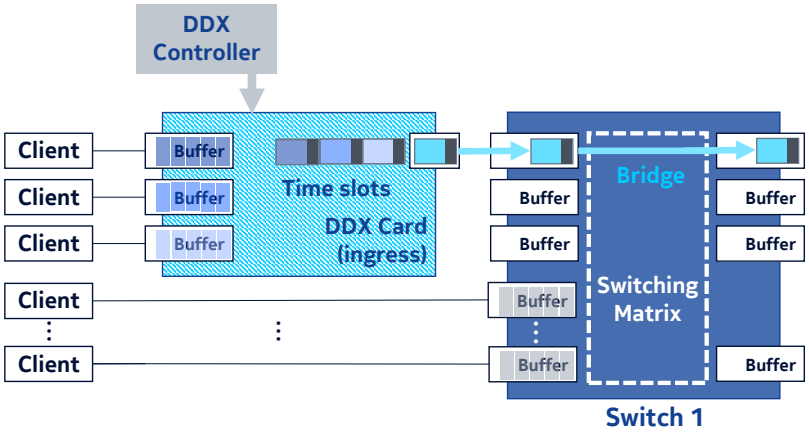

Figure 2.5: Time Division Multiplexing of traffic at first node insertion

The whole flow of DDX packets with the client data in their payload can then be sent onto the network that we bridged according to the protocol we described in the previous section.

2.4.1 Scheduling time-critical clients

We require the time-critical clients connected to a DDX node to send a request to the network orchestrator to be provided with a guaranteed latency by our DDX protocol. Similarly to SDN, the time-critical client sends a request to the orchestrator. The request must contain the source of the traffic, the destination, the throughput, and can contain the periodicity of the traffic. With this information, the orchestrator can compute a schedule to accommodate for this application.

In this manuscript we have not tackled the problem of scheduling as the literature has already expansively done so. Scheduling has long been demonstrated to be a NP-hard problem [29], and many algorithms have been proposed to tackle the problem in different circumstances. Part of the complexity comes from the sheer number of clients to organize at the same time. In order to limit the scaling of the problem, a spacial division approach has been proposed [30]. The network is divided into multiple TSN island where the traffic can be scheduled independently.

In the TSN community, the norm 802.1Qbv is being developed to allow for real time scheduled traffic. We can think of our DDX protocol as an enhancement of the TSN network, providing latency controlled link over 802.1Qbu preemption link, where clients would be scheduled according to the 802.1Qbv norm. Window-based schedules are being developed for the 802.1Qbv norm.

2.4.2 Reservation parameters

Given the application throughput, there are multiple parameters that we can tune to answer the application needs. The reservation is periodic. One period contains n slots. Each slot is attributed to at most one time-critical client to ensure client independence. The n slots form a window.

We are effectively dividing the bandwidth given on the deterministic path we have created into n time slots. Given the requested throughput, we can give one or more slots to a single time-critical application. We can choose to vary the size of the slots to accommodate the need of each application.

The periodicity of the window significantly influences the overall latency of time-critical traffic. For clients with a single reservation throughout the window, the latency between the ingress and egress node will be the sum of the propagation time throughout the network and a full window period. Due to the random arrival time in the ingress DDX node, a time-critical packet can arrive either exactly when its slot is reserved, when another slot is served or in the worst case when its slot is departing creating a full window waiting period in the ingress DDX node. Because we perform latency compensation for every packet to experience the same latency through the network, they all have to wait for the worst-case delay.

Having a periodic schedule is not well suited to bursty applications. The throughput of the bursty time-critical application might be low enough to be served by a single slot. But during a burst, the full line throughput might be required resulting in the traffic being divided into slots of successive windows resulting in a prolonged wait.

While most often time-critical traffic is not bursty, some application might use bursts of packets. For example, eCPRI has different modes of communication between antenna and base station. The mode ON/OFF sends burst of traffic. This mode allows for easier separation between antennas, as each antenna sends at once a high quantity of data but with a low refresh rate.

We do not think that our protocol is well suited for bursts of data, as TDM systems offer a poor use of resources to bursty clients. But this conclusion highly depends on the maximum throughput of the client and the throughput of the DDX bus. If the traffic sent during a complete window is able to fit within its allocated slots, then no issue will arise for a bursty time-critical client.

2.4.3 Flow independence

To ensure flow independence, a buffer is attributed to a single client. This ensures that no packet from an application will block the output of a different application.

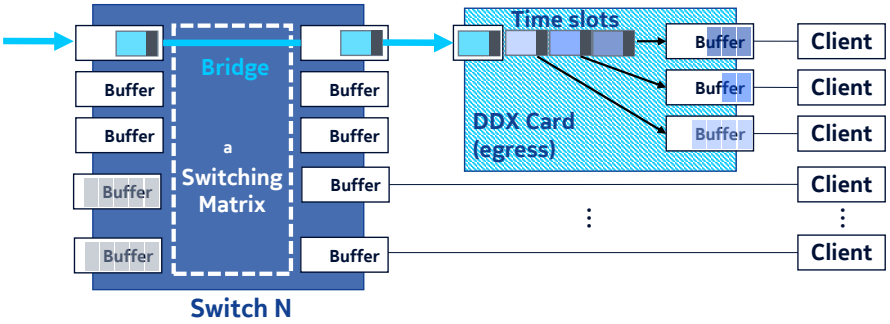

Figure 2.6: Clients are placed in independent buffer to ensure determinism

The information inside the slots is extracted from the DDX bus and placed into its dedicated buffer. In Figure 2.6, we show the distribution of clients at the bus egress.

2.5 Jitter compensation

With a deterministic path reservation through the network and the time multiplexing to handle possible collision between time-critical clients, jitter is already highly reduced compared to the applications free flowing through the network.

In this section, we first study the source of the latency variation for the packets at the output of the DDX bus, and then we present our method for providing theoretically zero latency variation.

2.5.1 Jitter source

We first need to mention that we are only able to mitigate the jitter between the ingress and egress device. Any latency variation happening outside our coverage cannot be estimated by our protocol. Hence, choosing to place our device below the top of rack switch to have a direct connection with the machine where time-critical applications run. If this placement is respected, only minimal jitter, few tens of nanoseconds, due to optoelectronics conversion will be introduced in the complete communication path.

The first source of jitter is the jitter introduced by the DDX protocol, more particularly the time multiplexing of clients. This source of jitter is easy to estimate,

we can store a counter with the time of arrival of a packet at the ingress DDX node, and account for this time at the egress DDX node.

Second, the deterministic path adds delay variation to the DDX bus. This source is unpredictable and the reason in the first place of this work. Depending on the type of bridge the amount of delay variation will change.

Creating a VLAN offers the less amount of variation, in the following experiments, we measured around 250 ns of delay variation for two switches. This jitter linearly increases with the number of switches on the path. The bridges created with the preemption technology showed a slightly increased latency variation. We measured a delay variation of 300 ns, of which 100 ns can be attributed to the non preemptable part of the packets. Leaving 200 ns introduced by the switch, showing that jitter has improved in newer products.

The latency variation is mostly due to the switch and not the preemption mechanism. Implementing a bridge over the network can impact other communications. As we presented in section2.3.2, we can increase the minimum threshold for preempting packets to let small packets pass through and not wait for the complete transmission of a DDX packet.

2.5.2 Ingress inter-packet delay recreation

The bridging techniques have allowed us to reduce the latency variation inside the network. The time multiplexing of time-critical clients has allowed us to prevent any contention between clients.

To reduce the remaining jitter, we perform latency variation compensation. We trade marginal latency for every packet to have the same latency. We provide the same latency for every packet of a flow by recreating the interpacket delay measured in the ingress node at the egress DDX node.

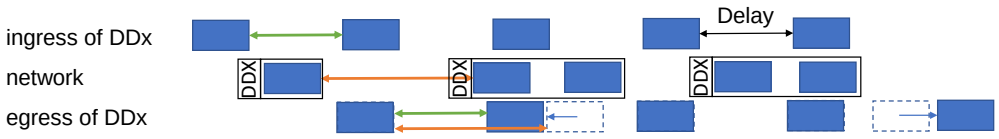

Figure 2.7: Achieving zero latency variation by recreating the ingress inter-packet delay.

In Figure 2.7, we show the process of the jitter compensation mechanism. In the ingress DDX node, we measure the gap between packets, and we send this information alongside the packet, as input to reconstruct the same delay in the egress

node. Due to latency variation on the deterministic path or during the insertion in the slot, the delay between packets may be completely altered.

In the egress DDX node, we send the first packet of the flow with an excess delay with respect to the second, corresponding to estimated upperbound of accumulated delay along the deterministic path. The following packets are sent after we have counted the number of clock cycles corresponding to the delay measured at the ingress.

By recreating the same delay between packets, we ensure that they all experience the same latency through the network. Theoretically this process can bring the latency variation down to a single clock cycle, 6.4 ns in our implementation. This system is able to fulfill the requirements of all the applications presented in Chapter 1.

2.5.3 Frequency offset estimation

In a hardware implementation, inter-packet delays are measured at one frequency and recreated at another. These frequencies are close, less than 100 ppm away but not equal. The frequency can lead to delay compression or worse to delay dilation creating a forever increasing latency.

Within Ethernet 802.3 this phenomenon is less of an issue as delays are not recreated. However, elastic buffers and inter-packet gaps are introduced to ensure that two network elements with different frequencies can communicate. These methods add jitter to the communication link.

In this section, we estimate the clock frequency difference between ingress and egress in real time in order to cancel any mismatch. We perform an estimate of the slope of the evolution of timestamps between ingress and egress and then virtually change the frequency of egress counting by adding cycles.

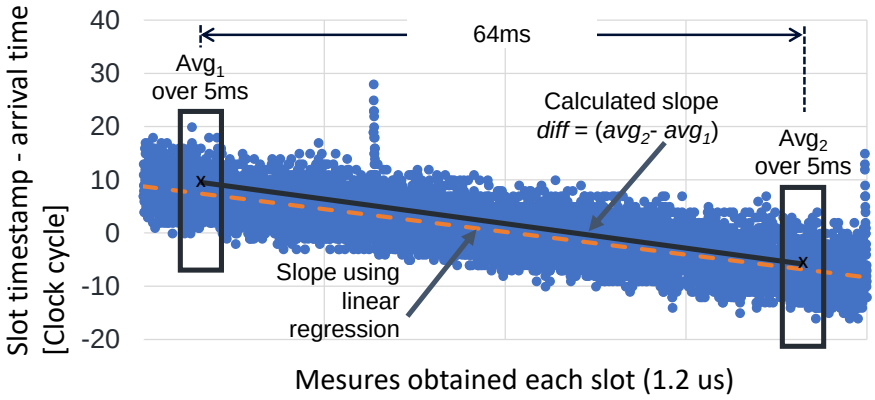

Figure 2.8: Estimation of the frequency difference between DDX nodes for the latency compensation mechanism.

In Figure 2.8, we present the estimation method for drift between two remote devices. We shall provide extensive details on the devices themselves later in this manuscript, but we compare here the rate of evolution of two counters, one generated at the ingress DDX node, and the other at the egress DDX node. Based on arrival time from samples recorded every 1.2µs, we can see with the blue dots the quasi-linear evolution of a time drift which bears, the signature of the ingress-egress clock mismatch that we are trying to estimate.

With each slot the ingress DDX node sends the value of the local time counter as information, partly to measure the time client packets have spent in the ingress device, and also to be used by the frequency offset estimator. The arrival time of the DDX packet is stored as the value of the egress node local time. We plot the difference of these two values in Fig 2.8. Because the two devices are not synchronized, the value of the difference is the sum of the deterministic path latency plus a random offset. As the latency varies, we can see that the difference is distributed over multiple clock cycles.

In the context of a deterministic path reservation, we can safely assume that the latency does not vary beyond the jitter present on the communication line. Any long-term evolution can therefore be attributed to a drift between the clocks of the remote devices.

In the Ethernet standards, the maximum frequency difference for a communication without fault is 100ppm [31]. All constructors of the communication must be compliant with this norm for communicating with 10G Ethernet. Yet, even though we use two identical development boards in our experiments, we can still record a drift between the local clocks.

We attribute this difference to hardware limitations. On the development board, a clock generator is driven by a quartz oscillator to generate the clock signal used by the Field Programmable Gate Array (FPGA). The Si5341 clock generator [32] has 90 fs RMS jitter and 0 ppm error. This component is extremely precise, the clock edges can shift in time with an extremely low margin, but the frequency recreated is the exact copy of the input.

On the other hand, the quartz used has a higher error margin. On the board, there is 48 MHz quartz [33] which has a stability of 10 ppm at room temperature. In the clock generator, the frequency is tripled and so is the error. Hence, with equipment of the same manufacturer we can have a frequency offset of 30 ppm. This corresponds to a drift of 30 µs per second. Hence, the need to correct it.

Assuming a slow variation of the frequency difference, we can estimate in real-time the frequency difference by performing a linear regression on the evolution of the time

offset between the ingress and egress node. In Fig 2.8, we show a simplified version of this estimation by counting the number of clock cycles of difference between two points separated in time.

In order to reduce the impact of the deterministic path jitter on the precision estimation, both points for the measure are the average of local samples. The points are separated by 64 ms, allowing us to detect a minimum of 1 clock cycle difference in 64 ms, corresponding to 0.1 ppm frequency difference. The samples shown in Fig 2.8 demonstrate an evolution of 16 clock cycles over 64 ms, corresponding to a 1.6 ppm frequency difference. The difference is below the maximum frequency error announced by the quartz constructor.

Knowing the ingress/egress frequency difference, we virtually adjust the egress frequency by periodically increasing or decreasing the egress counter to cancel the frequency difference.

2.6 Protocol implementation

In the previous sections, we presented the different elements combined to build the DDX protocol. With those elements we can theoretically achieve zero jitter, enabling the transfer of 5G protocols which require less 130 ns jitter, and ensuring that industrial machines can be operated from a remote server in an edge cloud server.

One of the main objective of this thesis is then to demonstrate in a real network scenario what performance can be reached. We first define which platform to build the DDX add-on card on.

2.6.1 Choosing the prototype platform

When building a prototype there are multiple criteria we need to observe. First, we must be on a real-time platform to ensure a deterministic processing of the clients. Second, the platform needs to operate down to the nanosecond scale to provide latency variation below a few hundreds of nanoseconds. Third, the platform needs to accommodate multiple clients.

With regard to these criteria, we chose to build our prototype on a Field Programmable Gate Array (FPGA) platform. The FPGA is an array of reprogrammable logic circuits which can be programmed to perform various tasks, such as controlling signals. We chose a Xilinx FPGA because they offered a dense and vast programmable space with many transceivers to connect to optical fibers pluggable.

While current microprocessors can run at frequencies above 3 GHz, current FPGA operate at frequencies around 200 MHz. But the structure of the logical arrays offers a high ability to run parallel tasks, which is well suited for the treatment of many parallel clients. The FPGA are able to send signal on optical fibers at frequencies over 100 GHz. These performances are highly sufficient for a prototype that has to communicate with servers, which most often use Ethernet 1G or 10G.

We can develop the logic to be fully deterministic and guarantee performance down to the clock cycle. With an operational frequency around 200 MHz, we can provide a precision of 5 ns. This level of precision is well below the requirements for the time-critical applications detailed in Chapter 1.

The FPGA is the logic core, they are often mounted, at least when prototyping, on development board which offer a high flexibility in the hardware that can be added. On the board we chose there are three FMC ports, where additional boards, called daughter boards, can be connected. On these ports 560 signals can be passed through. Daughter boards can serve a variety of functions, from memory storage to physical ports for optical fiber communications. We chose to have daughter boards providing 10 slots for pluggables, totaling thirty possible fiber optic connection to the FPGA. This number of clients is comparable to commercial switches.

In the following section, we detail the logical elements we programmed on the FPGA to build a functional prototype of the DDX add-on card.

2.6.2 Logic modules to build the DDX prototype

To build the DDX add-on card prototype, we need to program multiple functions in the FPGA logic. In Figure 2.9, we present the different logic modules that we implemented. We will describe them in the same order as the traffic from a time-critical application might experience them.

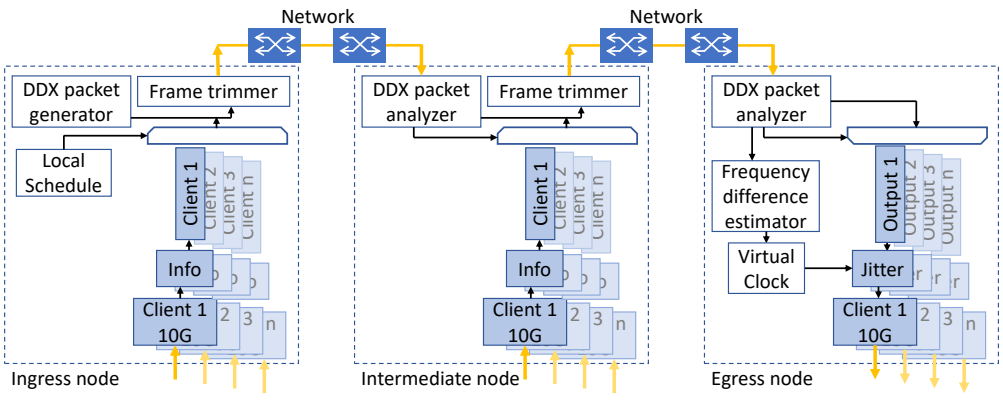

Figure 2.9: Functional blocs in the hardware implementation of the DDX nodes.

Multiple time-critical clients can be connected in parallel to one DDX card. Each client has its dedicated optical port. We configure the transceiver on the FPGA to connect a 10G Ethernet link. We first route the signals from the transceivers through two modules provided by Xilinx, the 10G Ethernet PCS/PMA and the 10G Ethernet MAC modules. The first module is in charge of the deserialization of the data and the clock recovery for the optical communication. The second module is in charge of the Ethernet protocol over the communication line. These modules perform the protocol part of the Ethernet over optical communication, and we as to user only provide the packet data.

The second step in the processing of the packets is to store information about each packet received to help transmit the packets over the DDX bus and to enable the jitter compensation in the egress DDX node. We interface the module named *Info* in Figure 2.9 and the 10G Ethernet module via an AXI4-Stream interface on which packets are sent. When receiving a packet, we save its time of arrival, the number of clock cycles separating it from the previous packet, and we measure the length of the packet. We then insert these three pieces of information in front of the packet. We implement this module for each client.

We then move the packet into a FIFO to store it before sending it on the DDX bus. We configure the depth of the FIFO to accept two windows worth of traffic at maximum bandwidth. In this implementation we use periodic windows of 8 slots, and slots are no larger than 1518-byte long. To allow for the processing of data of a 10G Ethernet client during the duration of two windows, we use a FIFO of width $64 + 8$ bits, 64 bits for data and 8 for control, and of depth of minimum $2 * 1518/8 * 8 = 3036$. $*2$ for the two windows worth of traffic, $1518/8$ as the data from a slot is stored in words of 64 bit width, and $*8$ as there are eight slots. Xilinx offers templates for FIFO and we used one of depth 4096.

Once the packets are stored, they wait for their reserved slots. To send the data of multiple clients from the ingress node to the egress node, we proceed to packet in packet insertion. We generate an Ethernet packet as long as the slot, in this implementation, 1518 bytes long. The source and destination MAC addresses of this packet are the ingress and egress DDX node respectively. Depending on the type of bridge we can use different Ethernet packet flavor. For prototyping we use the Ethertype 0x88B5 which represents a local experimental Ethernet type. We add a VLAN tag if one is used for bridging.

In DDX packet, after the Ethernet header, we add information about the slot. We add the time at which the slot is generated, and based on a configurable schedule, we add the identifier of a client to indicate from which queue to read. We then leave the

rest of the payload for the client frame. To our knowledge, frame embedding is not a common practice in Ethernet and is therefore not facilitated by the protocol. We need to choose an encapsulation method in order to be able to retrieve the beginning and end of client frames.

Here the main goal of the encapsulation method is to detect the beginning and end of a packet. In the Ethernet protocol, the encapsulation serves more purposes, like allowing clock recovery and data alignment. Common encapsulation method relies on encoding words of a certain length into a longer word length in order to reserve characters for flags as beginning or end of packet. This is a reliable method that provides an increased payload size. If words of 8 bits are encoded on 9 bits, an 8/9 encoding, the size of the payload is increased by a factor of 9/8. A different form of encoding is to use key characters. We declare for example that one 8 bit word is special and that the following 8 bit word should be interpreted as an instruction. Here the instruction would be a flag for the beginning of the packet and the end. But the key character can be found in the data of the packet, so there is an additional instruction that signifies that the key character was actually data. Because of this, the key character method does not yield a deterministic size increase.

In this prototype we use a more rudimentary method. To the client packet we added as a prefix the time of arrival and the length of the packet. With this information we know how much data to read in the egress node to retrieve the whole packet.

DDX packets had been generated as packets of length 1518 Bytes but might not have been completely filled. So after the client packets have been inserted we reduce the size of the DDX packet payload to match the size of the inserted packet. We perform this operation to avoid sending dummy data and leave usable bandwidth on the deterministic path with the preemption mechanism for exterior best-effort traffic.

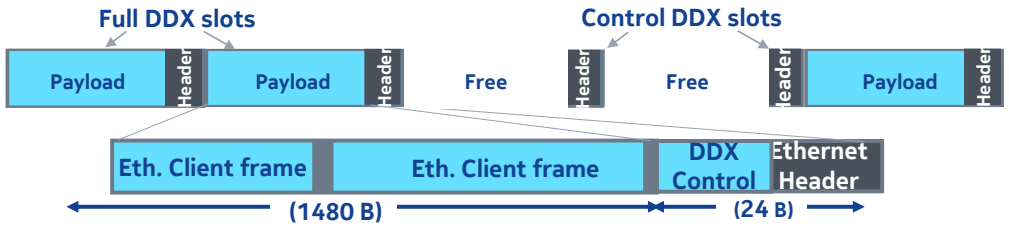

Figure 2.10: DDX bus frame construction

Figure 2.10 describes the DDX traffic on the deterministic path. It shows a DDX packet with the Ethernet header followed by DDX slot information, and the payload filled with client data. We show that even if no client data is ready to be sent the

DDX packet is still sent for timing purposes at egress. If no client data is inserted, we reduce the DDX packet sent to the Ethernet header, the control information and padding to reach the minimum Ethernet packet length.

To summarize, the DDX packets are built as follows:

Length	Value / Description
6 bytes	Destination MAC address
6 bytes	Source MAC address
(4 bytes)	VLAN tag (optional)
2 bytes	Ethertype, 0x88B5 : Experimental Ethernet
4 bits	Client id
20 bits	DDX Frame emission timestamp
39-1497 bytes	Payload

Table 2.1: Field description of the DDX Ethernet packet.

After being sent by the ingress DDX node the DDX packets are routed by the network equipment on the reserved path and reach either an intermediate node or the egress node. At an intermediate node, the control information is read, and traffic is inserted in the packet, in the same way as described above, if the slot is dedicated to a local client.

In the egress node, first the time of arrival of the DDX packet is recorded. We send the difference between the generation time of the slot and the time of arrival in the timing module to estimate the drift between clocks of the remote devices. Then the client payload is extracted from the slots according to the flags described in the encapsulation method. Each client is attributed a FIFO to maintain client independence. We then proceed with the jitter compensation mechanism to send client packets with minimal latency variation.

We reconstruct for every single client the delay between packets that we measured at the network ingress. The timing of each packet depends on the previous packet in the flow. For the first packet, which has no predecessor, we compute the emission time to ensure that the following packet will be in the FIFO when they need to be sent. We need the first packet to wait for the maximum latency variation that can be inserted on the path. With the timestamp of arrival at ingress and the slot timestamp we compute the time spent in the ingress node. Then with the time of the received slot we compute the remaining time to wait for a whole window. We

then add a margin to accommodate for the network jitter. The following packets are sent with respect to this first one. In this implementation we dedicated 20 bits for timestamps, which can measure up to 6 ms gaps between packets. If the gap between packets exceeds this threshold, we consider the next packet as the first of its flow and generate the latency based on its time of emission. To increase the maximum gap between packets, with can increase the number of bits allocated for the timestamps.

During the recreation of the time delays between packets, we proceed virtually modifying the local clock as described in section 2.5.3. This method enables a continuous delay recreation method without being affected by clock offset.

We presented the different elements that go into programming the DDX protocol over on an FPGA platform. From the reception and timing of time-critical clients to the framing and encapsulation, and finally the bus extraction and jitter compensation method. In the following section, we will evaluate how this implementation fares in a network with commercial products.

2.7 Experimental validation over commercial products

An objective of this thesis is to experimentally demonstrate our claims of providing ultra-low jitter in real network scenario. In this section we present the results achieved with our prototype to augment off-the-shelf switches.

2.7.1 Experimental setup

Having the partnership with Nokia for this thesis allows us to test our research over various network elements and technologies which can be seen in Figure 2.11. We demonstrate the performance of the DDX protocol in different realistic network scenarios. First we perform tests with classic switches by creating a VLAN. Then we test the impact of a shared bridge by using preemption, as described in TSN. For this experiment we configure the four switches presented on the right of the figure. Lastly we test the performance of inserting DDX add-on cards in an OTN network. On the left of the figure we can see two OTN node with aggregation cards and 1830 switches. At the bottom we can see two red boards, these HTG930 contain the FPGA on which we implement our protocol.

We measure the performance of up to eight traffic flows with different service level agreements, generated by a Spirent traffic generator analyzer. Four time-critical

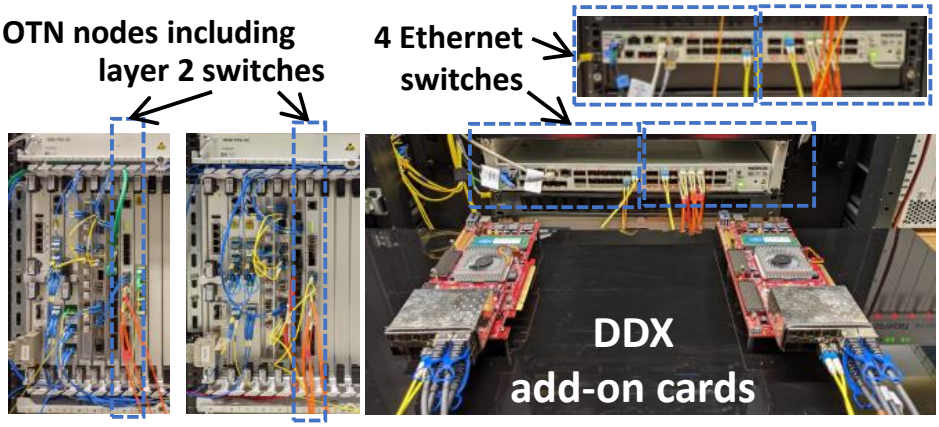

Figure 2.11: Experimental setup for testing the DDX concept in available market products.

flows of 1 Gbps throughput on a constant bit rate, and four best-effort flows of 1 Gbps throughput.

We run three different scenarios. First the traffic is routed through an OTN network. We measure the performance with and without the DDX protocol and add-on card.

Second, we make sure that data traffic travels across two switches. Up to eight clients emit eight constant bit-rate flows at 1 Gbps. These flows are aggregated in the first switch and transported over a 10 Gbps link, and disaggregated back. Even though the link throughput is greater than the sum of the eight individual clients, but some contention is observed. In a real scenario, the network link between switches would have a higher bit rate, but there would also be more clients resulting in contention. So we think that this experimental conditions offer a faithful representation of the concept even at higher bit-rates.

In this second experiment, we create a VLAN exclusively for the DDX packets. In this scenario as there is only one path doing a loop in the network, reserved in the VLAN, the best-effort traffic is lost, and therefore can cause no contention.

In the third scenario, we set up a bridge over four switches. We compare the performance of the time-critical frames being inserted in the DDX bus when the reservation is made with a VLAN and when the reservation is made with the preemption mechanism. When using preemption, the best-effort traffic can be sent through the network and causes some contention with the DDX packets.

2.7.2 Latency variation compensation over OTN network

We first show the latency distribution of time-critical traffic through a network segment based on OTN. Through this network we send two time-critical flow and one best-effort flow. We compare the measurements of the time-critical flows, in orange of the time-critical traffic passing through a 10 Gbps reservation in an OTN network; the traffic from the three sources are aggregated before entering the OTN node by a layer-two switch. In blue, the three flows are pre-aggregated inside the DDX node and sent to the switch to be inserted in the OTN network. The experimental set-up is detailed in the appendix OTN-shared.

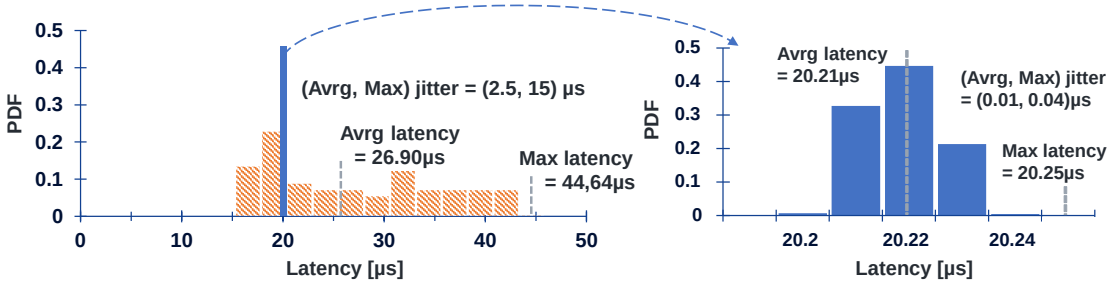

Figure 2.12: Latency distribution of time-critical traffic using OTN and a switch with and without DDX. Orange: without DDX, Blue: with DDX

In Figure 2.12, we can see in orange the low jitter proposed by the OTN with the container reservation. The maximum latency variation is about 30 µs. This latency variation can be attributed to two factors. First to the aggregation of the three flows in the layer-two switch before entering the OTN network. The second factor is the random wait for the allocated reservation in OTN.

In blue in Figure 2.12, we also report the distribution of the latency of the time-critical flows when the time-critical traffic is passed through DDX cards. We can notice an increase of the lower bound of the latency range, from 15 µs without DDX to 20 µs with DDX. However, this increase comes a strong decrease of the upper bound of the range, yielding jitter reduction by about three orders or magnitude. More specifically, on the right graph we display the distribution of latency of only traffic going through DDX. We observe a maximum latency variation no higher than 50 ns, and an average jitter of just 10 ns.

2.7.3 Latency variation compensation over TPS switches

In the second experiment, we show the improvement of the jitter experienced by four time-critical flows traveling through DDX and two Ethernet switches, when

compared to traveling to the switches only. In both cases, we create a bridge for the time-critical flows. We configure a VLAN and configure the Ethernet ports on the path to only accept traffic from this VLAN. This way, best-effort traffic cannot compete with time-critical flows. The experimental set-up is further described in the appendix 2S-VLAN.

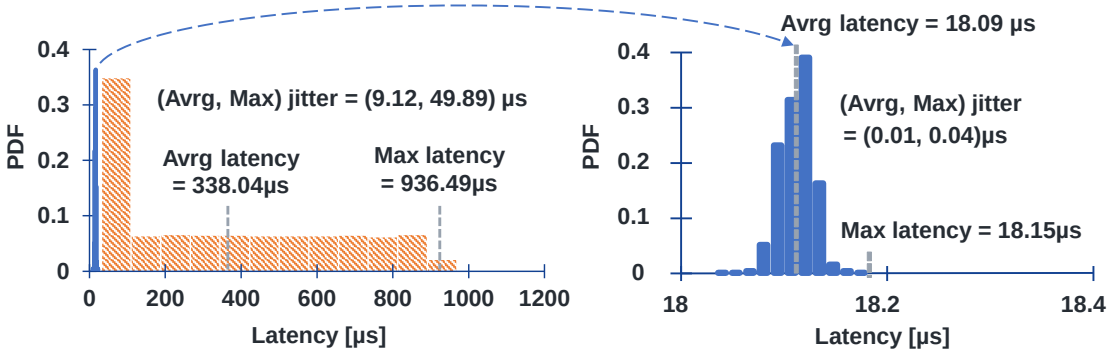

Figure 2.13: Latency distribution of time-critical traffic through a dedicated bridge in two Ethernet switches. Orange: without DDX, Blue: with DDX.

In orange in Figure 2.13, we represent the latency distribution of four time-critical flows. We can see the four flows competing for the resource resulting in a long tail distribution. This results in a maximum latency variation of just under one millisecond, with average jitter of 9.12 µs. These results apply only to one link between two switches. The jitter performance would naturally deteriorate with a greater and more realistic number of switches between source and destination.

In blue in Figure 2.13, we show the distribution of the latency of time-critical flows when going through the DDX nodes and the reserved path on the switches. Because the four clients are aggregated at the ingress DDX node to form a single data flow, there is no competition over the reserved path and the latency range is strongly decreased. Finally, with the jitter compensation implemented in our DDX, we further decrease this latency range, to achieve an average jitter of 10 ns, and a maximum latency variation of 160 ns.

With a bridge created by a VLAN, we completely eliminate any form of competition on the deterministic path. But we also dedicate one path to time-critical flows, that would otherwise be shared with best-effort traffic, while often over-dimensioning it. In Figure 2.14, we compare the latency distribution of time-critical flows going through DDX over a deterministic path reserved with a VLAN or through the preemption mechanism over four switches. The preemption mechanism allows for best-effort flows to go through the same path. In this experiment, four best-effort clients are

generating traffic going through the path as the time-critical flows. The experimental set-up with preemtion is further described in the appendix 4S-PREEMPT.

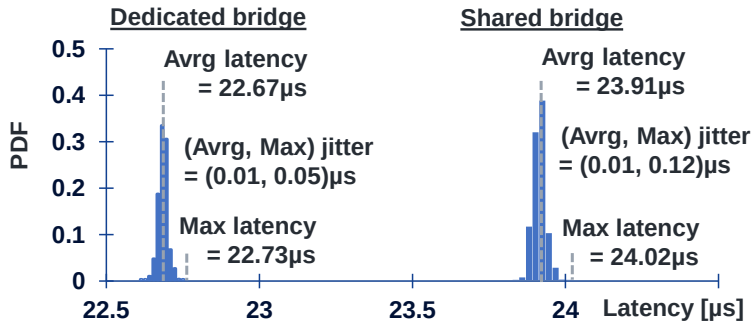

Figure 2.14: Experimental setup for testing the DDX concept in available market products.

In Figure 2.14 we demonstrate that with the competition of the best-effort traffic we are able to provide ultra-low jitter. The average latency is increased by 1.3 µs due to a longer processing in the switch. The average jitter is kept at 10 µs, but the distribution is more evenly distributed over the maximum latency variation. The maximum jitter measured by the Spirent is 120 µs.

The preemption mechanism offered by the TSN norm 802.1Qbu [28], allows us to reserve a path for time-critical traffic whilst allowing best-effort traffic to flow through. There is a marginal loss of performance, we are able to provide ultra-low jitter, with maximum jitter of 120 ns.

2.8 Conclusion

In this chapter, we introduced the DDX protocol to be implemented in stand add-on cards so that we can enhance legacy and future Ethernet networks to carry multiple time-critical packet flows.

We presented the three different key aspects for building the DDX protocol:

- **Engineering a deterministic path.** While legacy networks cannot provide timing guarantees to multiple clients, we are able to allocate resources for one client upon which we send all the data of the time-critical clients. We call a deterministic path such reservation where an upper bounded latency can be guaranteed. Such reservations can be made through different methods, through the creation and complete reservation of a VLAN to eliminate network competition. Or by leveraging network protocols developed by TSN as preemption which still allows for the transmission of nest effort traffic.

- **Multiplexing time-critical clients without contention.** The precedent step allows us to send the traffic over one controlled path. We need to eliminate the competition that could occur between the different time-critical clients. For this purpose, we employ a time division multiplexed bus where each client is given a reservation according to its requirements.

- **Compensating latency variation.** At the egress of the deterministic path jitter is accumulated for each client due to the random time of arrival at the ingress DDX node before being inserted in a dedicated slot, and the network jitter of the deterministic path. To reduce this latency variation, we implement a method to guarantee the same latency to every following packet by recreating the time delay between successive packets. To ensure the continuous operation of this process, we implement a mechanism to estimate the clock drift between remote devices to virtually modify the egress clock.

We performed an implementation on an FPGA hardware platform, and tested its performance with market available products. In a network with two or four switches we are able to provide 120 ns maximum RMS-jitter against competing best-effort traffic. We showed improved performance using complete path reservation using VLAN or OTN.

Scenario	Average RMS-jitter	Maximum RMS-jitter
OTN	10 ns	40 ns
VLAN over two switches	10 ns	40 ns
VLAN over four switches	10 ns	50 ns
Preemption over four switches	10 ns	120 ns

Table 2.2: Comparison of the DDX add-on card performance for different scenarios.

In table 2.2, we present the overall results of these experiments with commercial equipment. Without best-effort traffic contention, as demonstrated with the OTN and VLAN bridge we can reduce the maximum RMS-jitter to less than 50 ns. The increased number of switches which doubles the jitter on the deterministic path only causes a 10 ns increase in maximum jitter.

The performance we demonstrated could be useful to serve the most demanding applications of industry 4.0, and/or increasing the speed of distributed computing application. To meet the performance requirements for 5G communications, we need to demonstrate a timing alignment error maximum of 130 ns for class A eCPRI

traffic. Whatever the scenario where DDX is used, application designers will demand a proof that the latency guarantees are always met, for each and every packet, which traffic analyzers cannot provide.

In the following chapter, we propose and implement a monitoring system to measure the latency evolution and the timing error to show whether we are able to provide the performance required for 5G applications.

Chapter 3

Distributed Monitoring for time-Critical Networks

Contents

In the previous chapter we saw how to guarantee constant latency, which we can predict approximately from the deterministic path packets have taken. And we saw that extensive research efforts have been focused on designing network solutions for providing performance guarantee [8, 34–36]. Due to the multiplicity of flows from different qualities of service, time sensitive applications are more subjected to altered performance in a converged network than in a dedicated network.

In this context, a monitoring plane is crucial first to prove that the service level agreements are met per application. Second, some applications treating the network as a resource could leverage a better estimation of network metrics to lower scheduling margins for increased performances. [14]

In this chapter we will investigate which metric can be measured, then look at the existing solutions for monitoring. After we will take interest in the different synchronization methods necessary for distributed monitoring. We will present our prototype, its synchronization method and monitoring capability, and we will discuss the accuracy that we managed to achieve in an FPGA implementation.

3.1 Network performance to monitor

As networks grow in complexity, involving numerous devices, multiple segments, and various technologies, the need for reliable network operations and failure recovery becomes paramount. However, managing such diverse networks, especially with a wide range of technologies, poses significant challenges for Operations, Administration, and Maintenance (OAM) tasks.

Ethernet Operations Administration and Maintenance (OAM) [37] provides multiple functions :

- System or network fault indication
- Performance monitoring
- Security management
- Diagnostic functions

- Configuration and user provisioning

In this manuscript, our primary focus is on ensuring guaranteed latency in time-critical networks. Therefore, we will emphasize the performance monitoring aspect of OAM. Performance monitoring involves measuring various network parameters, including packet latency, packet loss, and maximum throughput. Given our objective of guaranteeing latency, our main focus will be on monitoring latency and its related parameters, such as jitter, in our experiments.

3.1.1 On monitoring latency

One way latency versus roundtrip time

When it comes to monitoring latency, there are two primary approaches: monitoring one-way latency and monitoring round-trip time.

- **Round-Trip Time (RTT):** This measures the latency from point A to point B and then back from point B to point A in a network. In other words, it calculates the time it takes for a packet to travel from its source to its destination and then return to the source.
- **One-Way Latency:** This focuses solely on the latency from point A to point B, measuring the time it takes for a packet to travel in one direction without considering the return journey.

The choice between these two approaches depends on the specific monitoring objectives and requirements of the network.

Monitoring round-trip time is indeed a straightforward approach to gain insights into a network behavior. By assuming symmetric latency, you can estimate one-way latency by dividing the round-trip time in half. This method is relatively simple to integrate into a network and can be implemented by having monitoring devices or services in various network locations sending probing packets to their neighboring devices. However, when it comes to monitoring the entire network, especially in complex and large-scale setups, challenges and limitations can arise, as will be discussed in the context of existing work.

Assuming symmetric latency across network links is often an oversimplification, especially in complex distributed networks where traffic patterns and loads can vary widely in different directions. For time-critical applications, monitoring each link direction independently can indeed be crucial to gain a more accurate understanding of network performance. This approach allows you to identify potential

bottlenecks, delays, or issues specific to each direction of communication, enabling better optimization and management of network resources.

Monitoring one-way latency in a network is a more complex endeavor compared to monitoring round-trip time. It necessitates monitoring endpoints at both ends of the communication path, and these endpoints must be synchronized to provide accurate and precise latency measurements. The synchronization precision is indeed a critical factor in determining the overall precision and reliability of a distributed monitoring system for one-way latency measurement. Achieving precise synchronization across network elements is a challenging but essential aspect of accurate performance monitoring in time-critical applications.

Application or network point of view

The measured latency can be seen from two angles. Either from the application or the network point of view.

Applications that require deterministic latency need to monitor the complete end-to-end latency of their data path to ensure their requirements are met. A failure to meet these requirements can often lead to at best performance losses and at worst safety issues endangering equipment or even life. Network operators and managers often focus on individual link latency and network segment performance. This allows them to identify bottlenecks, optimize routing, and ensure efficient network operation. In the context of a deterministic network with guaranteed latency, the network orchestrator needs to monitor and manage the latency for the specific applications it is managing.

In this manuscript, will focus on monitoring one way latency over the full path that the applications take in order to verify whether their performance demands are met.

Different metrics for latency

When monitoring latency, the network operator might look for different metrics that each require different monitoring and processing methods. The low-effort metrics will be the average, the minimum and maximum latency of a path. They require the least effort to set up as only a few memory cells and a few logic cells to compute the average will be necessary. Though these measures do not provide much insight on the behavior of a network path. A network operator, based on this information might set up an alarm to the maximum latency, that when it surpasses a threshold, some action will be performed.

A second approach, often employed in experiments to understand the network behavior, is to represent latency using a histogram. This method demands additional logic and memory resources. It also requires the network operator to specify the width of each bucket that forms the histogram. This tool can be precise when defining buckets of low timespans and lack an ensemble view, or contrary provide an ensemble view but gather many packets of different latency in the same buckets.

The most complete method is to monitor the latency of every packet. This method allows us to recreate the statistics of the two previous methods but requires a lot of memory space depending on how much data the network operator is willing to store. Monitoring every packet latency provides the ability to identify trends in the network behavior, making it a valuable input for machine learning algorithms designed for network monitoring.

While the other methods can rely on probing packets to perform measurements, timing every packet requires information to be sent with each packet. Inserting a timestamp inside a packet becomes the preferred solution, as having a probing packet following each application packet would significantly reduce the usable throughput.

In this manuscript, we will build a prototype to monitor the latency of every packet and propose various methods of result reporting. This includes presenting the data as a histogram, where the width of each bucket must be specified, or as a graph illustrating the latency evolution over time. Considering limited buffer space, users can choose to observe short-term changes by monitoring the latency of the past N packets or opt for long-term monitoring by measuring every n packet to gain insight into its evolution.

3.2 Existing work

Monitoring has been widely used in offline networks, employing test devices to evaluate network performance. In our laboratory, we utilize a Spirent traffic analyzer as a reference to generate traffic and analyze various metrics. This analyzer is equipped with two cards, each featuring four ports capable of generating traffic ranging from 10 Gb/s to 100 Gb/s. It can measure latency distribution with a precision of up to 10 nanoseconds, as well as assess maximum throughput, traffic loss, and other metrics. The traffic it generates is fully configurable, allowing us to test various network technologies with customizable protocols and payloads.

While a Spirent analyzer is a valuable tool for network characterization, it is limited to offline network assessment. This limitation arises because the analyzer needs to function as both the source and destination of the packets it generates,

requiring it to loop back onto itself for network performance measurement. This constraint prevents its use to provide real time monitoring over hot network to enable feedback on the performance provided.

In-band Network Telemetry (INT) [38] has been developed to gather various network characteristics inside reprogrammable networks. To achieve this, it builds upon the Programming Protocol-independent Packet Processors (P4) (Programming Protocol-Independent Packet Processors) language, which was designed to open up the software controlling network equipment, allowing network controllers to define their own behavior through the programming of specific rules.

The P4 language facilitates the operation on the header fields of the packets. INT was proposed as an extension of P4 and defined a telemetry field that could then be used directly by the switch.

In the context of detecting potential network faults related to applications generating micro-bursts of traffic (lasting from 10 to 100 microseconds), Joshi et al. introduced a concept called "burst radar" [39]. This burst radar, as described in their work, aims to identify and monitor these micro-bursts, which can significantly impact network latency. To ensure a decent quality of service for other applications, it is important to take appropriate measures to control the applications responsible for these bursts.

The proposed approach focuses on collecting information about the packets that make up the micro-bursts. By analyzing the packet-level data, the burst radar can identify the specific applications that are causing these bursts. This information is valuable for network orchestrators, as it allows them to take targeted actions to address the issue and optimize network performance.

Network monitoring has recently been developed in order to automatize the network tuning in the context of SDN. The TSN family of network technologies which guarantees delay performance for time sensitive traffic [40] is also developing solutions to monitor latency, to report the given performances and enable traffic management based on network performance.

3.2.1 SDN monitoring solutions

The SDN community has defined a paradigm to perform packet-based communication with a data plane for communications and a control plane for accessing all the network equipments. Having constant access to all the network equipment allow to fine-tune the network. Therefore, being able to collect the network characteristics becomes a great advantage for an efficient network.

By leveraging the control channel and a link discovery protocol, Liao et al. [41] are able to monitor one-way latency between the different network equipments. They propose to insert timestamps in the Link Layer Discovery Protocol (LLDP) packets and program the switch so that it updates the timestamps of those packets on its way. At the same time, the controller can estimate the latency to the switch by halving a round-trip time measurement. And subtract the latency to the switch to the round trip latency of the LLDP packet giving the one way latency of the link.

They reach 5% measure precision with a low over head as they leverage packet that are already present in the network. Howerver, the update of the LLDP is usually on a period of more than 5 s, usually 30 s. Thus, if they want to increase the refresh rate of the monitoring they need to increase the frequency of LLDP, increasing the network overhead.

Atary et al. [42] present how the leverage the SDN protocols to send packets along selected path to measure round trip latency. The user may select path to monitor. Their system will configure two flows to propagate from the server source of the measure to the targeted path. By having one flow stopping at the ingress of the path and the other at the egress and both returning to the source server on the same path they are able to measure the latency of the selected path.

In order to optimize the overhead of their method, a single probing packet is sent from the source server and duplicated at ingress path. Furthermore, they apply search algorithm over the tree like architecture of networks to minimize the number of packets to be sent to cover all the links to be monitored.

The precision of the system increase with the depth of the measured links, The standard deviation of the measurement is around 0.5 ms. Because they need to configure the path through the SDN network, the setup phase takes 2.7 ms due to the SDN protocols that are often slow.

In the context of SDN, monitoring methods often rely on probing packets and measuring round-trip time. But in order to collect reliable information on the latency of time critical packets, direct packet monitoring becomes essential. Furthermore, when aiming to perform one-way latency measurements, it is necessary to deploy multiple monitoring endpoints distributed throughout the network with a way to understand their timing information.

3.3 Synchronization protocol for distributed networks

Most systems either measure round trip time or estimate one-way latency by halving the round trip time. In offline setups, equipment can measure one-way latency by acting as both the source and destination of the traffic. However, in online networks, two devices are required: one at the network ingress and one at the egress. To ensure precision in the measurement, both devices must have synchronized time.

Throughout history, various synchronization protocols have been proposed and implemented to enhance the precision and stability of time-sharing synchronization.

3.3.1 Network Time Protocol

The Network Time Protocol (NTP), initially formalized in 1985 in the RFC 958 norm [43] provides a synchronization protocol and infrastructure globally available aiming to achieve sub-second synchronization precision.

The current version of NTP, described in the norm RFC 5905 [44], includes specifications for the network architecture, the protocol for message exchange between devices, and the algorithms required to perform synchronization.

It will be common throughout the study of synchronization protocols and in our project that the network is logically organized in a tree-like architecture. This structure establishes a clear hierarchy for organizing different endpoints within the network. Servers connected to high precision clock or atomic clocks are categorized as *stratum 1*, and the *stratum* of another server represents the minimum number of synchronization links separating it from a *stratum* 1 server.

Time synchronization is mostly distributed vertically, and a server will choose among its neighbors with the lowest *stratum*, which is considered to have a better quality, to perform the synchronization. A horizontal synchronization, between two servers of the same *stratum*, is sometimes performed. It increases the precision of the synchronization between this pair of devices and provides redundancy in case a parent server is unreachable.

The synchronization protocol operates through the exchange of timestamped messages, allowing the computation of latency between two servers and the offset between their clocks.

The round-trip latency is given by the formula:

$$L = (T2 - T1) - (T'2 - T'1)$$

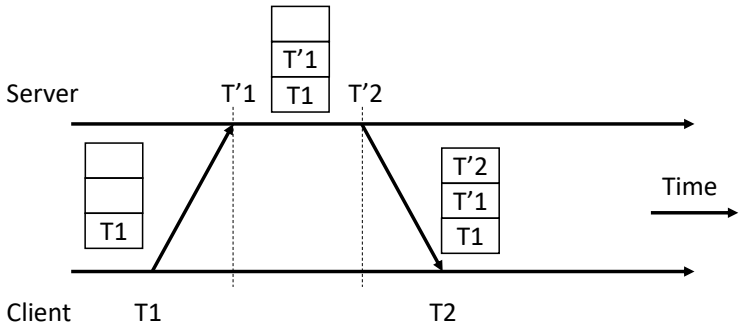

Figure 3.1: Timestamp exchange in the Network Time Protocol.

As synchronization is most often performed over a direct link, halving this quantity yields the one-way latency with a decent precision.

The offset between clocks is given by:

$$\Theta = \frac{(T'1 + T'2) - (T1 + T2)}{2}$$

This formula is strictly accurate when the one-way latency is equal in both directions. However, if the one-way latency is not equal, the offset will have an error equal to half the difference of the one-way latencies. This underscores the significance of having a symmetric path for accurate synchronization.

These formulas serve as the foundation for time synchronization, and the precision with which one can estimate these values determines the accuracy of the synchronization.

3.3.2 Precision Timing Protocol

The PTP was first formalized in 2002 in the norm IEEE 1588 [22]. It was developed to improve on the NTP protocol and reach a synchronization precision on the microsecond scale. Nowadays, it is widely used for various applications, including synchronizing financial transactions and mobile phone towers.

To enhance precision, the PTP employs several techniques. First, it duplicates timestamped messages to convey the precise time when the first packet left the server, reducing the uncertainty caused by network client processing delays. Second, if the PTP communication occurs through compatible switching equipment, information about the time spent in the switching equipment is added, helping to limit the uncertainty introduced by latency variations in the network. Further improvements of PTP allow to timestamp the packets directly in hardware. These measures collectively contribute to improve the synchronization precision.

In a ideal environment, assuming zero delay variation and link symmetry, the precision of PTP is limited by the resolution of the timestamps it communicates or the smallest increment at which those timestamps are updated.

3.3.3 SyncE

Both the NTP and PTP protocols ensure that the clocks in different devices are synchronized at a specific point in time. However, after the initial synchronization, the clocks evolve independently. Ideally, both clocks should tick at the same frequency and remain synchronized over time. In practice, various factors like temperature, power supply difference, and others can cause the oscillators to operate slightly differently. This leads to a drift in the clocks, which accumulates over time and results in desynchronization.

To mitigate this issue, the quality of oscillators is carefully controlled to limit the frequency difference, thus limiting the drift. In 10 Gb/s Ethernet, the frequency difference for oscillators limited to a 100 ppm difference [31]. For devices operating at a frequency of 156.25 MHz, this frequency difference can result in a drift of 100 µs every second. The systems therefore operate multiple synchronizations per second to prevent the clocks from drifting too far apart.

The objective of Synchronous Ethernet (SyncE) is to minimize the frequency difference between different network elements.

First the accuracy of the local oscillator of a SyncE equipment in free-running conditions should not be greater than 4.6 ppm with respect to a reference clock. Thus limiting the potential drift to 4.6 µs every second.

Secondly, and more importantly, SyncE defines a network architecture to propagate a clock frequency from a source clock to all other devices. In a tree architecture a synchronization signal is passed down along the branches thus limiting the possible drift between devices and setting up a perfect environment for time synchronization. After frequency synchronization the drift between devices should be kept under 40 ns per seconds [45].

3.3.4 White Rabbit

While networks benefits from nanoseconds to microseconds precision monitoring for providing feedback to time critical applications and enabling a cooperation between application scheduling and network scheduling. Other domain of science require a tighter precision. The European Organization for Nuclear Research, CERN,

has developed the project White rabbit to achieve sub-nanoseconds to picoseconds precision of synchronization between thousands of scientific equipments.

The sub-nanoseconds precision is achieved by compiling the methods above.

First frequency synchronization is performed using SyncE. Then PTP is used to convey latency and offset between devices. Detailed link characteristics, obtained through measurements during setup and continuous monitoring, help account for any asymmetry in the communication path And lastly, phase frequency detectors are used to detect the phase difference between the master and slave clock, enabling to improve the resolution of PTP limited to the clock period.

The White Rabbit protocol demonstrates which precision can be reached. However, for monitoring packet network a sub-nanosecond precision is not necessary. Packets are processed words by words, where words consist of 8 to 512 bits depending on the Ethernet speed. Usually the speed of treatment of those words is at a word per 5 nanoseconds. For 10Gb/s Ethernet, words are composed of 64 bits and are processed with a period of 6.4 ns. Thus, the synchronization precision we are trying to reach for monitoring Ethernet packet is around the nanosecond scale.

3.4 Proposed distributed monitoring device

We introduce a monitoring plane designed to provide precise (in the nanosecond scale) per-packet latency measurements for optical networks. This monitoring infrastructure comprises standalone devices placed along specific network paths for monitoring purposes. In the following sections, we provide a detailed description of our proposed monitoring plane and its implementation. Furthermore, we assess the accuracy of synchronization and the precision of per-packet monitoring measurements.

In our baseline use case, as previously described, we assume that time-sensitive traffic represents a relatively small portion of the total network traffic. To ensure economic viability, it is essential to deploy time-sensitive-grade equipment only where necessary. As a result, we have designed our monitoring plane to be modular. Our approach involves placing multiple nodes at the ingress and egress points of the paths that require monitoring.

In Figure 3.2, we present a schematic of a typical converged network scenario where certain application flows are time-sensitive and require deterministic paths, thus necessitating monitoring. These flows are often generated by machines, which we refer to as “servers” in this context, though they could also include interfaces for robots on a factory floor. In the case of a data center, for instance, we would position the monitoring devices between servers and the top-of-rack switch. This

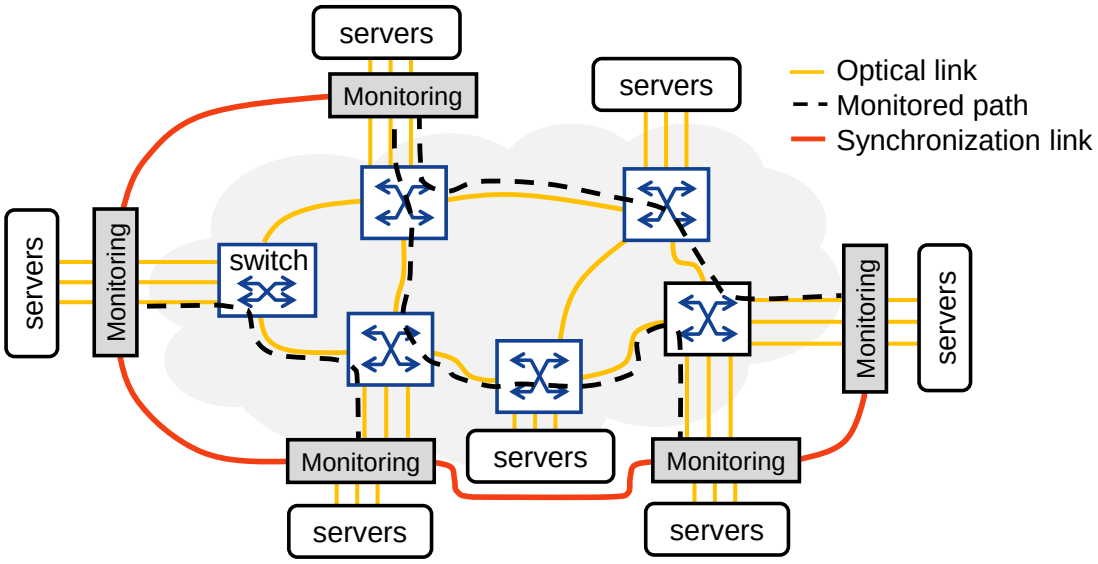

Figure 3.2: Monitoring device placement in network.

configuration allows us to measure the latency between endpoints over a path with variable latency.

In order to achieve a precision under a hundred nanoseconds, we need a precise synchronization of time. The synchronization is first limited by the network over which it is run. Therefore, we require a network with exceptionally low jitter since any uncertainty in synchronization directly translates into uncertainty in our measurements. However, because our goal is to measure the latency over the DDX network we cannot use the prototype presented before to carry this synchronization flow as any bias over this link would be found in the synchronization process without any means to detect it.

We therefore chose to have our own network architecture for providing the synchronization as represented in red in Figure 3.2. We deploy monitoring devices in a tree architecture to perform master-slave synchronization. Each device is interconnected to the next via a dedicated optical fiber, referred to as synchronization segment in the following. When concatenated, segments and devices form a synchronization link. Similar to the concept of NTP, a reference clock is passed down through a hierarchy of these devices. We employ frequency tuning to ensure persistent synchronization.

3.4.1 Metrics captured

Positioning the monitoring points along the path of the packets grants us access to a multitude of metrics. Among these, we are able to measure latency, jitter, client data rate, and loss percentage. In this thesis, our primary focus is on ensuring

consistent travel time for time-critical clients. Therefore, in our implementation of the monitoring device, we have concentrated our efforts on measuring latency and jitter.

In order to measure latency a timestamp is inserted in the monitored packet at the ingress node and extracted at the egress node. Application can be chosen to be monitored or not and once one needs to be monitored, its flow identifier, composed of source and destination MAC addresses, is sent to the monitoring points so that only these packets have a timestamp inserted. The monitoring points are transparent to other traffic flowing through.

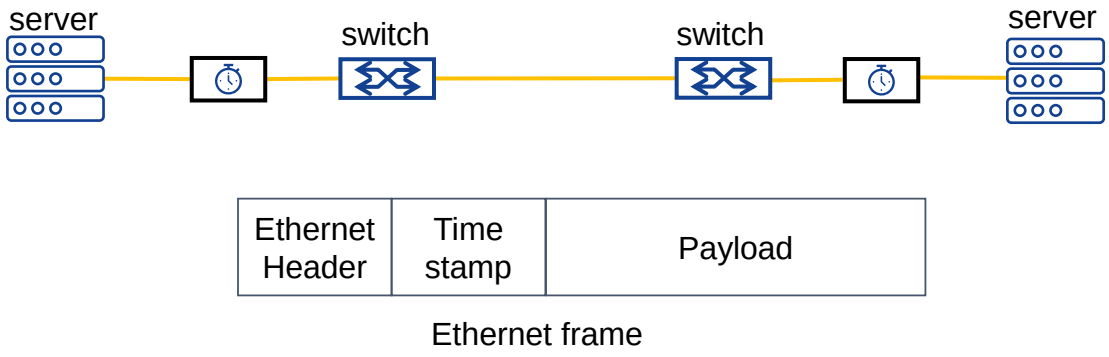

Figure 3.3: Time stamp insertion in Ethernet packet over network.

To maintain transparency within the network, we have adopted the in-band network telemetry approach, as outlined in [46]. This approach designates a reserved space behind the packet header to accommodate a timestamp. In our specific implementation, we allocate 32 bits for the timestamp. This 32-bit timestamp can be utilized unambiguously to measure timespans on the order of tens of seconds (e.g., in our experiment described below $2^{32} * T_0$ where T_0=6.4 ns is the clock period corresponds to 15.5 s.). This timespan is vastly over the average latency we expect to measure in data centers and metro-networks.

3.4.2 Our synchronization strategy

To ensure precise latency computations, it is imperative that all monitoring devices share a common time reference. To achieve this, all devices are interconnected via a dedicated network that serves dual purposes: synchronization and metrics reporting. These monitoring devices are strategically distributed in a tree topology on this network.

Similar to established synchronization protocols like NTP, we designate a master node, which may optionally be connected to a Global Positioning System (GPS)

clock, although this is not mandatory. The critical requirement is that time remains consistent across all devices. Subsequently, the devices synchronize themselves following a master-slave protocol, where the originating node propagates its time to its neighboring nodes and so forth.

To facilitate the exchange of time information, we employ PTP, which is capable of achieving synchronization down to the nanosecond level, provided that the synchronization segment is well-defined. This choice underscores the importance of our dedicated network for synchronization

In our architecture, the downward communication link comprises the TX interface of the master, the optical fiber, and the RX interface of the slave. The return way of the synchronization messages is the exact opposite, TX of the slave, optical fiber and RX of the master. By sending through the same fiber, we ensure that the travel time in each direction is equal, or that the difference between both directions is inferior to a nanosecond, which we will not be able to detect. The symmetric nature of the synchronization path allows us to have an accurate measurement of the offset and latency between both devices, following PTP computation of timestamps.

In order to increase the quality of the synchronization, we perform frequency matching (syntonization) between neighboring devices. We retrieve the master clock frequency from the Phase Lock Loop (PLL) of the slave device receiver and use it to tune the Digitally Controlled Oscillator (DCO) of the slave device transmitter. After syntonization, the master-slave clock frequency mismatch is essentially determined by the accuracy of the DCO, i.e., 10 part per billion for the clock generator of our development board, and the precision at which we are able to measure the master frequency.

We employ the method of syntonization, involving the reconfiguration of the DCO. This approach differs from using SyncE, which directly relies on the retrieved frequency from the communication link. Our rationale for this choice is to ensure that even in the event of a link failure where communication frequency would be lost, the mismatch between the master and slave clocks remains contained. This containment is possible because the local oscillator has been configured to the matching frequency, creating a time margin for reconfiguring the system in case of a link failure.

The syntonization works as follows:

1. On board quartz generates periodic signal.

2. Clock generator increases the quartz frequency to 156.25 MHz.

3. The FPGA uses the clock signal for Ethernet optical communication.

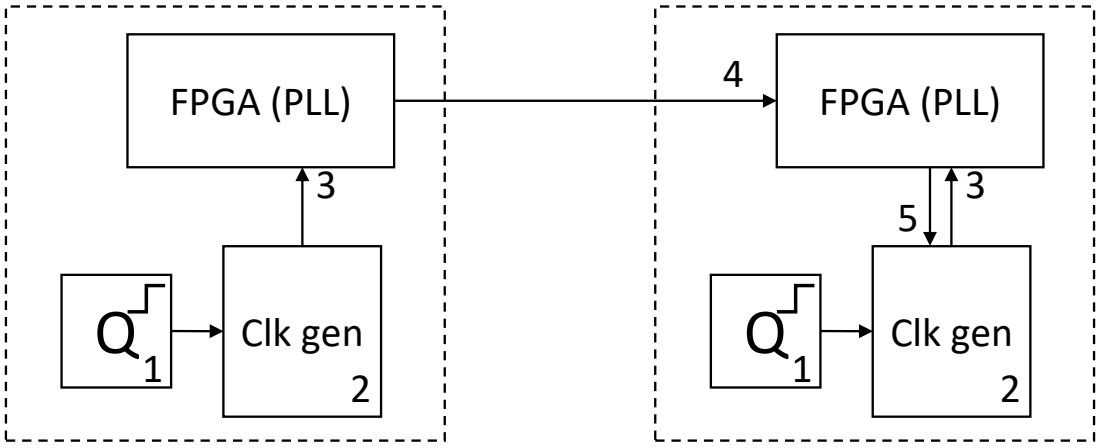

Figure 3.4: Synchronization procedure between monitoring devices.

4. The clock signal is sampled at the receiver FPGA.

5. The received clock is used to tune the local oscillator.

On the slave FPGA, we have implemented a module designed to compare the received frequency with the local oscillator frequency. This module periodically provides the count of clock periods that differ within a given timespan. Once we have this information, we can determine which clock is running faster and calculate the necessary adjustment for our local oscillator

In our implementation, we have selected a comparison period of one second. This configuration enables us to detect even a one-clock period difference over 156,250,000 periods, which equates to a tuning precision of 6.4 parts per billion (ppb). The precision of the DCO utilized on our evaluation board is well-suited to achieve this level of tuning accuracy.

3.4.3 Synchronization accuracy

We have devised a synchronization protocol enabling a global understanding of transmitted timestamp. The precision of our measurements solely relies on the precision of the synchronization process. In this section, we experimentally assess the precision of the synchronization in our implementation of the monitoring device.

To assess the precision of synchronization, we established a chain consisting of four devices, incorporating three synchronization segments. Leveraging the clock architecture on the development board we utilized, we were able to implement two monitoring devices in each FPGA. The HTG930 development board features two independent clock generators, each powered by its own quartz oscillator.

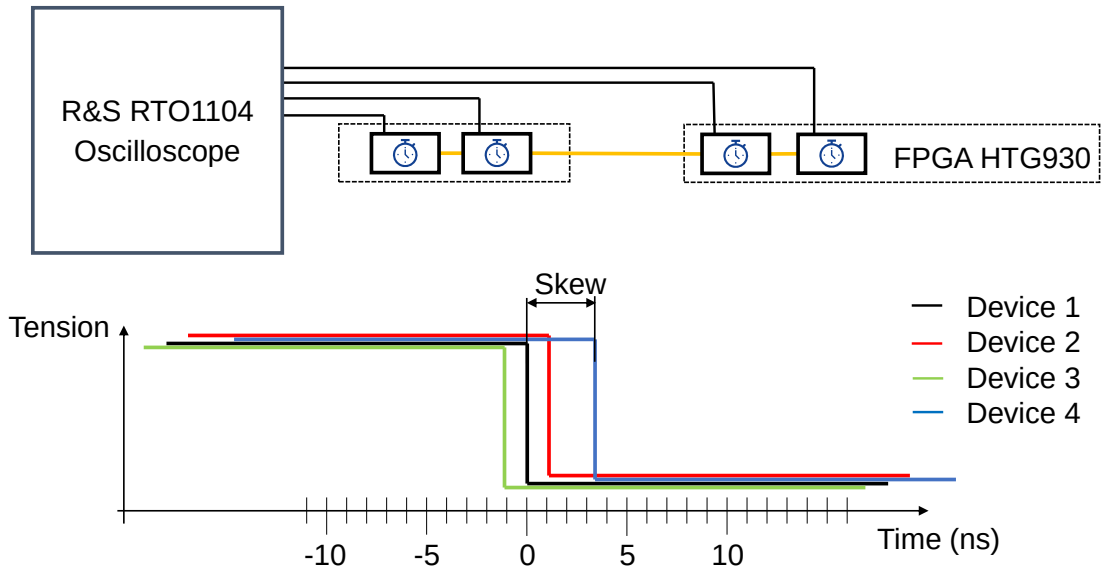

Figure 3.5: Synchronization measurement experimental setup.

Each monitoring device generates a rectangular waveform on a port connected to an oscilloscope, which monitors all four signals. We conducted comparisons based on the timing of the falling edges of these signals. In this setup, device 1 served as the source of synchronization (master) and passed this synchronization down to devices 2, 3, and finally 4. Each signal could be compared to the signal produced by device 1, used as a reference, to determine time-dependent skew. The rectangular waveform generated by the monitoring device operated at either 1Hz or 1kHz, depending on whether we were monitoring long-term or short-term evolution of the synchronization.

Importance of syntonization

In our initial experiment, we explore the necessity of syntonization in two distinct scenarios. In the first case, as depicted in the upper half of Figure 3.6, we illustrate the skew between the first and second devices, which are separated by just one synchronization segment (one hop). In the second case, as shown in the lower half, we demonstrate the skew between the first and fourth devices, separated by three successive synchronization segments (three hops).

The figure is further divided horizontally into four sections, each depicting the evolution of the skew over periods of 32 seconds with samples taken every millisecond. We define “wander” as the difference between the maximum and minimum values of the skew within a sliding window of 32 seconds. In the first scenario, only time synchronization is implemented with a low refresh rate of 1.2Hz. In the second scenario, there is again only time synchronization but with a greater refresh rate of

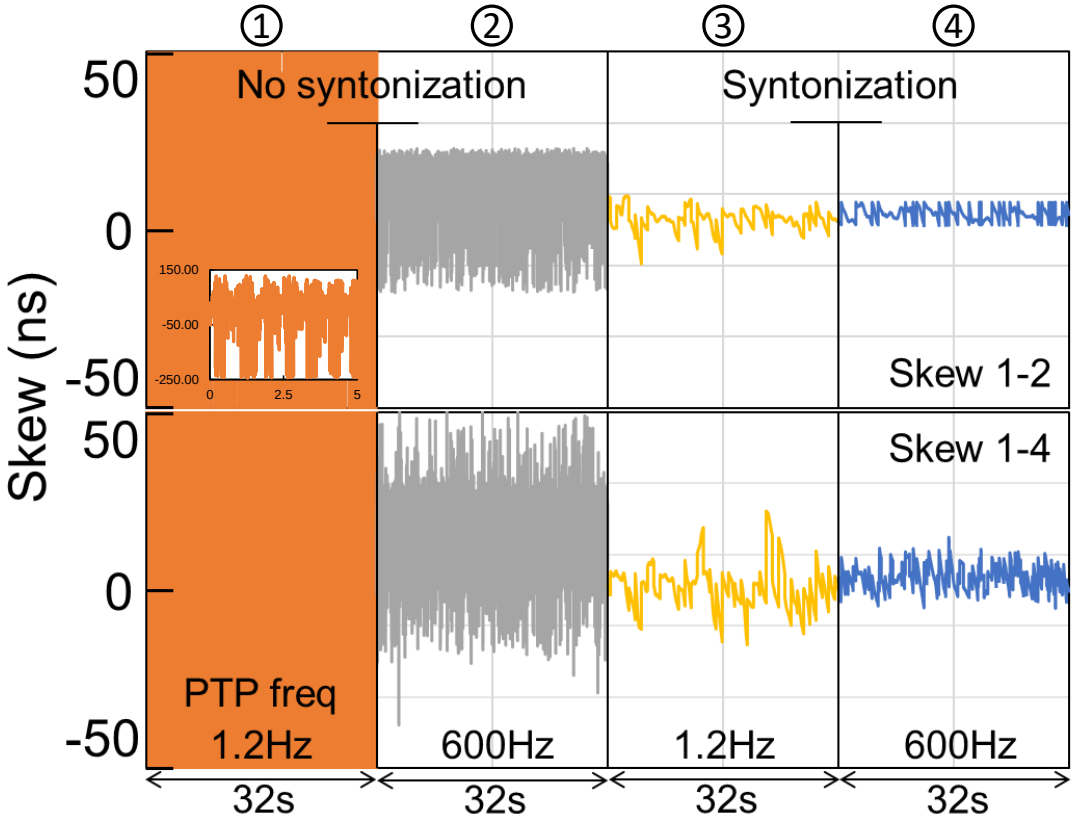

Figure 3.6: Evolution of synchronization over 32 second periods.

600Hz. In the third and fourth scenarios, time synchronization is refreshed at 1.2Hz and 600Hz, but frequency tuning is also enabled.

In the first scenario, the measurements exceeded the scale for both links, 1 hop and 3 hops. For the 1-hop scenario, we measure a 370 ns wander. This wander occurs within a second and is subsequently reduced with each synchronization event. In the inset, we provide a zoomed-out view of the graph, focusing on the first five seconds of the experiment. Over the 32-second duration, we can observe 38 peaks in a periodic pattern, here six peaks over 5 seconds, corresponding to the maximum drift that occurred before each synchronization. Although the development cards feature the same model of quartz and clock generator, with similar configurations, a 0.37µs wander was still observed. This discrepancy could be attributed to factors such as temperature variations within the cards or power management. Nevertheless, even with this slight discrepancy, the setups exhibit a difference of only 0.37ppm, which is remarkably low for communication devices. By comparison, Ethernet devices can experience frequency discrepancies as high as 100ppm.

In the case of 3 hops, the standard deviation of the wander remains at approximately 0.37µs, and we observe the same 38 peaks. Notably, the wander does not

increase, thanks to the synchronization protocol in place. The synchronizations are initiated sequentially, with the first device synchronizing with the second, and once the second device is synchronized, it initiates synchronization with the third device, and so on. During the brief moments when synchronization occurs, only minimal wander can occur. However, after synchronization, both devices tend to drift apart in a similar uncontrolled manner, much like devices separated by only one synchronization segment.

In the case of time synchronization with a refresh rate of 1.2Hz and no syntonization, precision is inherently limited by clock drift. Even with identical equipment and chips on the same evaluation board, minor frequency differences can arise due to variations in temperature or power supply, leading to synchronization imprecision. This variability in frequency difference results in synchronization precision that fluctuates significantly over time, making it unsuitable for monitoring purposes. To address this issue, we initially increased the refresh rate of the synchronization process. In the second scenario depicted in Figure 3.6, we can observe a reduction in maximum wander. For one hop, the wander over 32 seconds is limited to 40 nanoseconds. The shorter resynchronization period helps to control the wander. However, in the case of three hops, the maximum wander is slightly higher at 90 nanoseconds. Here, the number of resynchronizations begins to impact precision because the precision of time synchronization becomes a factor at this scale.

The precision of PTP is primarily determined by the resolution of the timestamp. In this implementation, we use the same clock for timestamping as the one driving the Ethernet communication interfaces, operating at 156.25MHz, which provides a resolution of 6.4ns.

To further enhance synchronization quality, as shown in Figure 3.6, syntonization proves to be essential. In the third and fourth scenarios, we maintain the same synchronization refresh rates as before but add the feature of tuning all devices in a sequential chain to the frequency of the first device. This results in smoother, more continuous curves, indicating a slower evolution of the drift between devices. It demonstrates the effectiveness of syntonization in improving synchronization.

When implementing frequency tuning and a high synchronization refresh rate we can see that the wander is kept under the minimum precision achievable, a clock period. Under this threshold we cannot with these methods affect the synchronization, leading to the observed skew fluctuating within this interval.

However, in the case of three hops, the maximum skew is slightly greater, approximately three clock periods. This decrease in precision is a result of adding the

imprecision introduced by the three links in the chain. We will delve deeper into this phenomenon as we examine the long-term distribution of the skew between devices.

Figure 3.6 illustrates that frequency tuning plays a pivotal role in stabilizing the synchronization process. Furthermore, a higher refresh rate of time synchronization proves effective in maintaining accurate synchronization by limiting the drift of clocks.

Evolution of the synchronization with the number of synchronization segments

In this second experiment we show the evolution of the synchronization accuracy by analyzing the distribution of skew between devices over a duration of 15 hours. The experimental set-up is the same as the previous, with four devices connected in a daisy chain configuration, with each segment propagating time and frequency synchronization to the next.

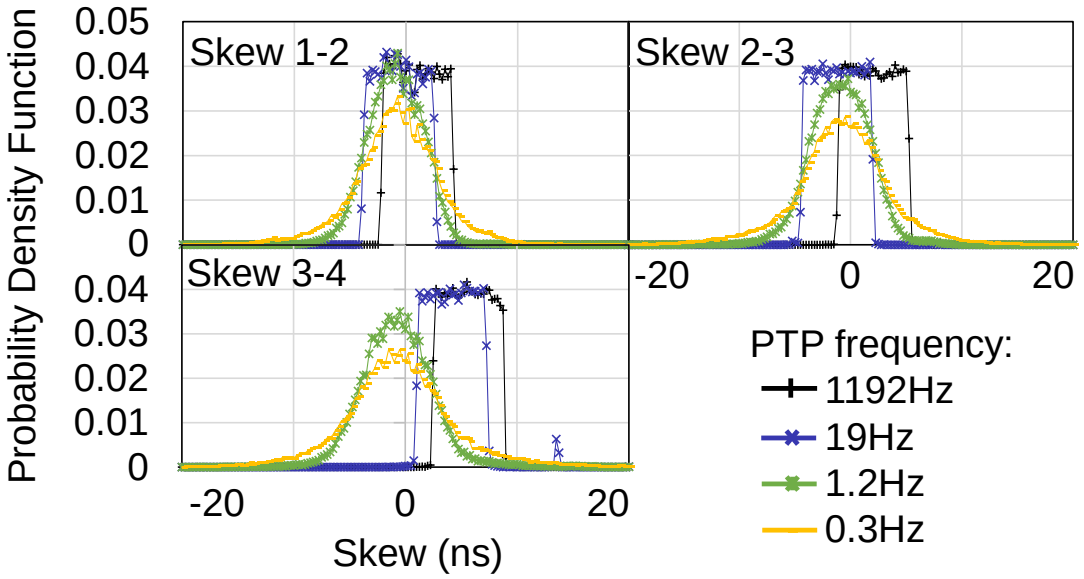

Figure 3.7: Long-term (15h) distribution of skews between different devices separated by 1 synchronization segment.

Figure 3.7, we present the long-term distribution, comprising 65,536 samples taken at approximately one sample per second over a 15-hour period. This distribution illustrates the skew between pairs of directly connected devices, involving only a single synchronization link in these measurements. We compare time synchronization refresh rate, 1192 Hz, 19 Hz, 1.2 Hz and 0.3 Hz, corresponding to periods of 2^{17}, 2^{23}, 2^{27} and 2^{29} consecutive 6.4 ns clock cycles, respectively.

For the highest refresh rates, the skew follows a square distribution with a width of 6.4ns, indicating that clock drifts occur within at most one clock cycle. This configuration represents the minimum precision achievable through these techniques. It is worth noting that the distributions are not centered on zero skew. We attribute this deviation to the fact that the back-and-forth travel time in the considered segment varies by a few nanoseconds. Minor differences in routing within the FPGA for the TX and RX could introduce a small imbalance, causing the equation of PTP, which assumes symmetry, to displace the position for zero skew.

Interestingly, despite using the same cards and fibers for all measurements and potentially encountering lab temperature variations over consecutive days, the skew distributions for different PTP frequencies on the same hop are not centered around the same value. We conjecture that this offset primarily arises from the limitations of the syntonization process and varies depending on the initial mismatch between clock frequencies at the start of the process.

Additionally, in our particular implementation, we employ elastic buffers to facilitate the transition between the clock domains of the FPGA. These buffers operate autonomously and may occasionally introduce uncertainties, such as skipping one clock cycle in rare situations. Addressing these uncertainties would necessitate the implementation of a supervisory mechanism for buffer operation, a task that we did not pursue in this manuscript

As the PTP refresh rate decreases, we observe an increase in the standard deviation of the skew distribution. While the devices are frequency synchronized, it is important to note that this synchronization does not involve the same clock being replicated throughout the entire chain, as would be the case in SyncE [45]. Consequently, the devices gradually drift off-center, in an unpredictable manner, leading to a normal distribution.

On the last link of the chain, between devices three and four, we can observe that a portion of the distribution is situated 6.4 ns beyond the end of the square distribution for the PTP refresh rate of 19 Hz. This behavior could be attributed to the resynchronization process, where the device ahead in the chain changes the cycle it is synchronized to, while the device further down the chain has not yet made the transition. This results in a full cycle difference, hence the presence of a peak located 6.4 ns away from the distribution.

In summary, for one synchronization segment, our use of frequency matching allows us to achieve the minimum skew, which is equal to one clock period, here 6.4 ns. We can reach this precision by fully controlling the path on which the synchronization

communication is operated, hence choosing to have a parallel network dedicated to synchronization and monitoring reporting.

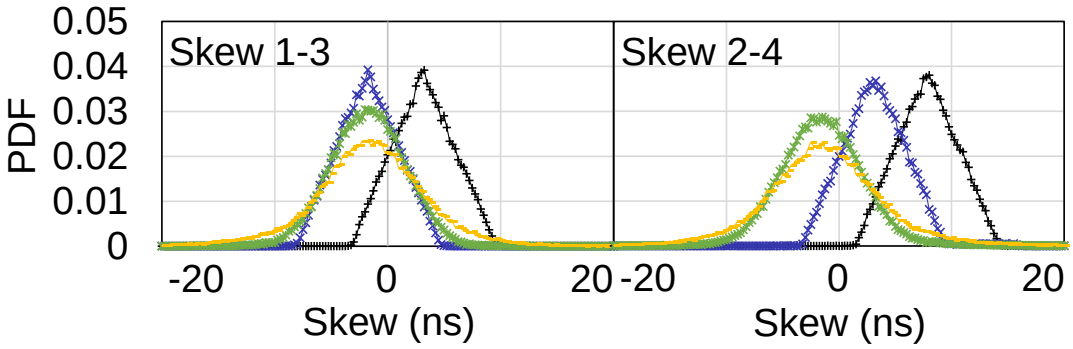

Figure 3.8: Long-term (15h) distribution of skews between different devices separated by 2 synchronization segments.

In Figure 3.8, we present the long-term distribution of the skew between pairs of devices separated by two synchronization segments. Given our chain of four devices, we now examine the skew between device 1 and 3, as well as between device 2 and 4. Notably, the distribution with a high PTP refresh rate exhibits a triangular shape with a base of 12.8 ns. This distribution is formed by taking the convolution of the two links composing the examined path.

We observe once again that the distributions are not centered around zero, which is a direct consequence of the one-hop synchronization not being centered around zero. Notably, the offset for one refresh rate is the sum of the offsets for the same refresh rate within the distribution for one synchronization segment. For instance, consider the center of the distribution with a refresh rate of 1192 Hz for the skew between devices one and three, as seen in Figure 3.8, which is at 3.3 ns. This value is the sum of the offsets for the distribution in Figure 3.7 with a refresh rate of 1192 Hz for links between devices one and two (offset of 1.1 ns) and between device two and three (offset of 2.1 ns). By adding these two offsets together, we obtain the offset for the composition of the two links. This phenomenon is consistent across all other links and refresh rates, providing us with valuable insights to develop a model for predicting the precision of synchronization for an arbitrary number of resynchronizations.

We realize with this first observation of the skew distribution that the skew distribution of one whole link is the convolution of all the individual links. Indeed, the convolution of the two square distribution is a triangular distribution.

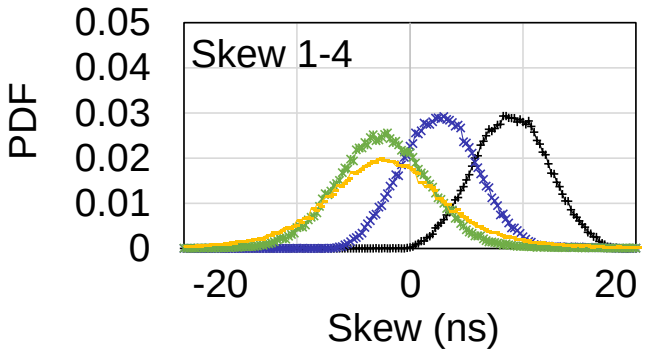

Figure 3.9: Long-term (15h) distribution of skews between different devices separated by 3 synchronization segments.

In Figure 3.9, we observe the skew distribution for a link composed of three resynchronizations. Here, we once again witness the same effect, where the skew distribution for three links is the result of convoluting the three individual links. Convoluting three square distributions yields a distribution with slopes that are second-degree polynomials, and the center is the sum of the means of all the individual distributions. Notably, the width of the distribution for the highest synchronization refresh rate is 19 ns. The high refresh rate ensures that the skew between the first and fourth device remains within 3 clock periods. However, as the refresh rate decreases, the width and standard deviation of the distribution increase. For instance, with a refresh rate of 1.2 Hz, the width is approximately 37 ns, extending further from the three clock periods.

This evolution, observed with varying numbers of segments, conforms to an Irwin-Hall distribution. Consequently, we can predict the uncertainty associated with multiple measurements. Given that the synchronization refresh rate ensures a square distribution of skew between devices within a single segment, the skew between devices in a link will follow an Irwin-Hall distribution with a parameter equal to the number of segments within the link. We can estimate both the mean and standard deviation of these measurements. Specifically, the standard deviation of skew for a link composed of n segments is calculated as $\frac{6.4*n}{12}$ ns.

In this section we measured the precision of the synchronization which is used to insert timestamps in the packets to measure latency. The precision of the latency measurements hence depends on the synchronization precision. We found that thanks to syntonization, time synchronization can produce a synchronization precise to the nanosecond scale, limited by an uncertainty of one hardware clock period.

The syntonization enables decreasing the time synchronization refresh rate. Here, we chose to perform syntonization by reconfiguring on the fly the clock generator of the monitoring devices in the chain. We introduced this method for "safety" reasons. Once configured, the clocks will continue to beat at the same frequency even in the event of a link failure in the monitoring network.

The precision of the syntonization here is given by the measurement of the difference of frequency in the syntonization protocol. Each second, we compare the number of clock cycle between the local clock and the received clock, and we tune the local oscillator given the difference in number of clock cycle. Measuring the difference of clock cycle each second grants for clocks beating at 156.25 MHz a precision of 6.4 ppb, getting close to the maximum precision at which we can tune the oscillator on the board 1 ppb.

Given the precision of the syntonization, the time synchronization is stable with an uncertainty of one clock period, here 6.4 ns. The skew between two devices therefore follows a uniform distribution over this clock period. We are unable to predict the center of the distribution and presented multiple factors that could explain the placement of the center. We expect the center of the distribution to be within one clock period of the zero-skew mark. We can make this assumption because of the attention we placed in performing the time synchronization and making sure that the PTP exchange occurs on a symmetric path. The White Rabbit [47] project further increased the precision of the synchronization by having a better knowledge of the synchronization link. During the setup of the WR devices, the propagation time is measured, the difference in propagation time due to different wavelength accounted for. This enables an improvement of synchronization.

To cover more ground and monitor more flows, multiple synchronization segments are placed in a chain. The precision of the synchronization decreases the more segments in the chain there is but follows a precise law. The concatenation of two links results in the convolution of their precision distribution. As the distribution of a single segment is uniform, the distribution of n segments follows an Irwin-Hall distribution of parameter n. This ensures a precision of $n * 6.4$ ns with a standard deviation of $\frac{6.4*n}{12}$ ns. With such synchronization precision we will be able to monitor latency precisely, in line with the latency guarantee we are aiming to provide the DDX card.

3.5 Experimental evaluation of the monitoring devices' performance

In the previous section we studied the precision of the synchronization between devices of a chain of one to four devices. The synchronized clocks are used to timestamp packets that need to be monitored. In this chapter we will compare the precision of the monitoring devices to a reference and examine whether the precision achieved in synchronization is effectively transferred to the accuracy of our measurements.

In our experimental setup, we compare our device to a traffic generator and analyzer used as a reference. Our Spirent equipment is capable of generating up to 32 parallel 10 Gb/s Ethernet traffic flows. For this experiment, we utilize six of these flows to transmit and receive data across six distinct paths that connect all possible pairs of monitoring devices, selected from our pool of four devices. The traffic analyzer functions by looping back the egress to the ingress with a remarkable 10 ns resolution, providing the minimum granularity for latency samples. While this setup makes it an exceptional standard for benchmarking against our monitoring devices, it is important to note that it may not be suitable for real network configurations where ingress and egress nodes are not looped back to the same location and it does not operate on live traffic.

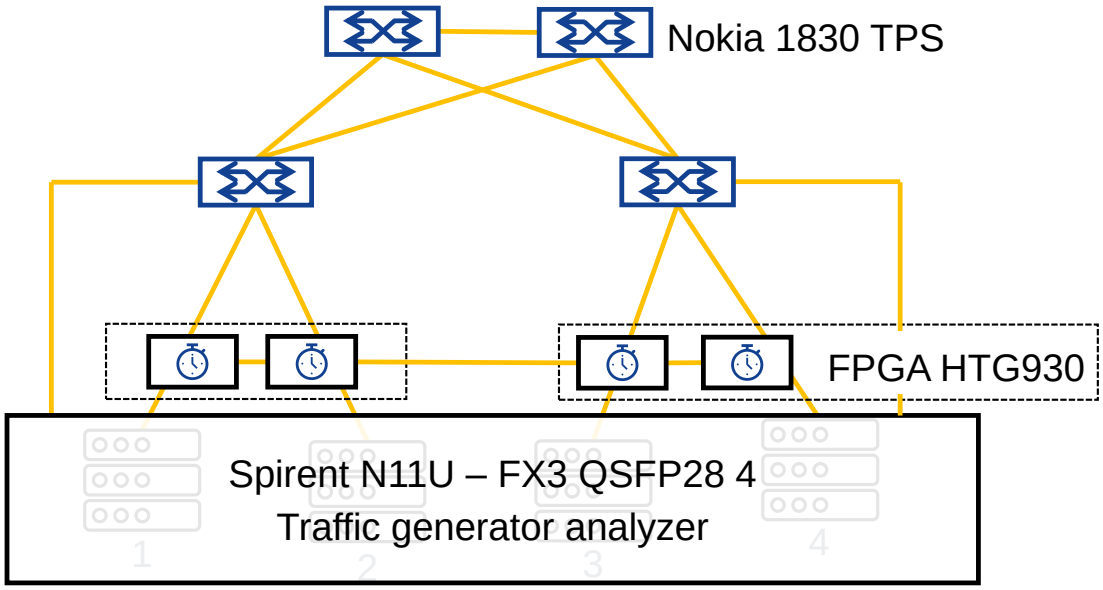

Figure 3.10: Experimental setup for measuring monitoring precision

In Figure 3.10, we provide a description of our experimental setup. In practical scenarios, the monitoring system finds its application within data centers to observe latency between pairs of servers or in industrial settings to measure communication delays between machines. In our setup, we substitute the servers with a Spirent test equipment that is able to generate traffic and measure its latency. We connect our four monitoring devices to the Spirent with 10 Gb/s Ethernet connection over optical fiber. The monitoring devices are then independently connected to a mock-up network consisting of four switches forming the base of a Spine-leaf topology which is often found in data centers. To simulate realistic network conditions, these switches are fed not only with monitored traffic but also with best-effort traffic, competing traffic. Best effort traffic is provided by other ports of the Spirent test equipment.

Unless specified otherwise, the traffic flowing through the monitoring devices consists of constant bit rate (CBR) traffic, characterized by 256-byte packets at a rate of 2.5 Gb/s. CBR traffic is commonly employed in time-critical communication scenarios, especially when stringent safety requirements are in place. In contrast, for the competing traffic, we utilize longer 1412-byte packets to emulate best-effort traffic, which resembles typical video streaming scenarios.

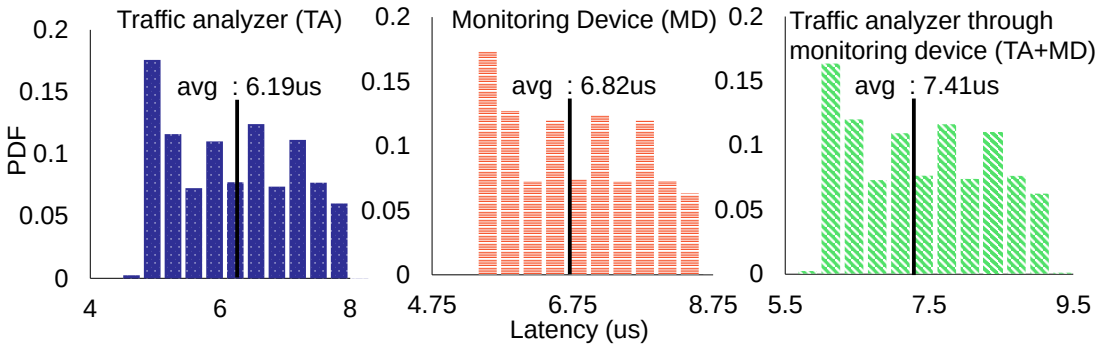

Figure 3.11: Latency distribution measured by the traffic analyzer, monitoring device and both

In our first experiment, we compare three measurements of traffic latency. Figure 3.11 presents these comparisons. In the leftmost graph, we depict the Probability Density Function (PDF) of latency measured with the traffic analyzer when directly connecting the Spirent test equipment to the switch after removing our monitoring devices from the network. The middle graph illustrates the PDF of latency measured by our monitoring system, using the network configuration described in Figure 3.10. Lastly, the rightmost graph displays the PDF of latency measured by the traffic analyzer when the traffic passes through the monitoring device, as outlined in Figure 3.10

Upon comparing these three PDF, we can initially observe the latency introduced by our devices. The insertion of an electrical device into the optical path results in a slowdown of traffic. Specifically, we notice that placing the monitoring device in the optical path leads to an increased latency of 1.22 µs. This latency increase is attributed to the processes of deserialization, serialization of data, and timestamping.

The added latency can be divided into these different sections:

1. Deserialization at the ingress monitoring device. Performed in the FPGA by a module created by Xilinx [48, 49] taking on average 33 clock cycles.

2. Timestamping and processing through FIFOs, taking a minimum of 14 clock cycles.

3. Serialization in the ingress monitoring device to be sent in the network. Performed in the FPGA by a Xilinx module taking on average 34 clock cycles.

4. Travel through the network on the path to be monitored.

5. Deserialization at the egress monitoring device. Performed in the FPGA by a module created by Xilinx taking on average 33 clock cycles.

6. Timestamp extraction and processing through FIFOs, taking a minimum of 14 clock cycles.

7. Serialization in the ingress monitoring device to be sent in the network. Performed in the FPGA by a Xilinx module taking on average 34 clock cycles.

Inserting a timestamp into the packet increases its length, particularly if the packet did not originally include a dedicated timestamp field. As the packet traverses the network, it encounters various network devices that typically process packets in a "store and forward" manner. This processing involves temporarily storing the packet until fully received before forwarding it, which results in a slight increase in latency if the packets are longer, typically by a clock cycle for each store operation. This phenomenon partially accounts for the observed difference between the estimated added latency and the actual added latency.

The insertion and extraction of the timestamp occur midway through the monitoring device. Consequently, the measured latency includes the latency added by the monitoring device, which encompasses steps 2 to 6. To obtain results equivalent to those of the traffic analyzer, we can subtract the known added latency from these measurements.

In summary, Figure 3.11 demonstrates that our monitoring system enables latency measurements that deviate by less than 1% compared to the traffic analyzer measurements, with only a minor additional latency of 1.2µs

All the above assume constant bit-rate traffic. Should traffic be bursty, a small load-dependent latency can be added by the ingress monitoring device. We study next how to circumvent this issue by carefully tuning the interpacket gap. The latency of the packets might be increased during the communication in the network elements due to the store and forward process taking one more clock cycle with the time stamp inserted. Extending the length of the packet will cause another issue to appear only when the throughput reaches its maximum, here at data rate of 10 Gb/s. This problem can arise during bursts of traffic, when for a short period of time the throughput reaches its maximum. In Figure 3.12 we examine the evolution of latency of packets in three consecutive bursts of 50 packets

Impacts on latency of the monitoring device during bursts of traffic

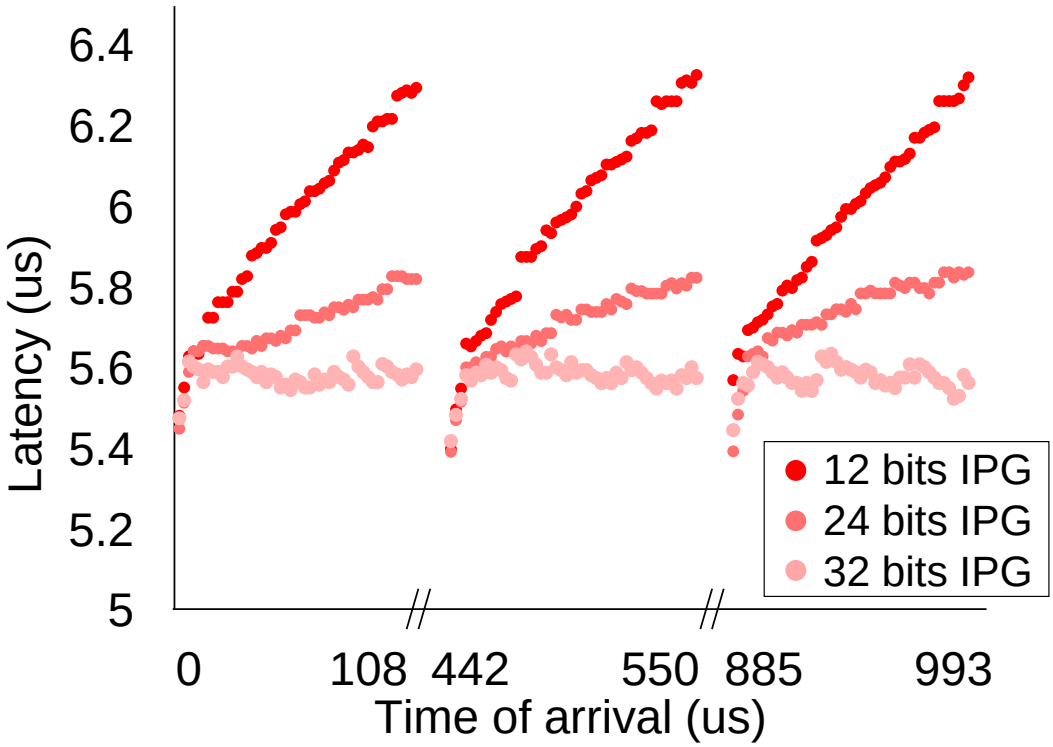

Figure 3.12: Latency evolution of 50 packets bursts for 12, 24 and 32 bits of minimum interpacket gap (IPG).

In Figure 3.12 we can see the evolution of latency during three consecutive bursts. The bursts last for approximately a 100 µs, and we set the data rate to the line

maximum, here 10 Gb/s. To accommodate for clock frequency mismatch, Ethernet protocol requires a minimal gap between packets of 12 bits [31]. The network operator can choose to increase this minimum interpacket gap to accommodate its needs. On Figure 3.12 we compare the evolution of latency for three different interpacket gaps, 12, 24 and 32 bits. We configured the traffic generator to produce bursts with these interpacket gaps for the network accepts any gaps higher than 12 bits.

Due to our insertion of information inside packets, their length increases. Furthermore, we process the packets at the same frequency as they arrive, adding one clock cycle to the processing of each packet. During a burst, the packets are streamed almost continuously through our user logic, so the added clock period accumulates throughout the entire burst of packets.

Here is how this effect manifests:

- The first packet flows through the ingress monitoring device with no added wait.
- The second packet waits for one extra clock cycle compared to the minimum waiting time.
- The third packet waits for two additional clock cycles, and so on.
- The n*th* packet waits for n-1 clock cycles.

This effect is clearly demonstrated in Figure 3.12, where we observe the latency increasing throughout the burst. However, it is essential to note that this latency increase resets after the burst, and the subsequent group of packets will have the same minimum latency.

The behavior of different interpacket gaps reveals an interesting trade-off. Because packets can be sent with a minimum gap of 12 bits through network devices, configuring a burst with a higher gap allows us to trade interpacket gap duration for space inside the packet. This effectively limits the increase in latency during bursts. However, it is worth noting that the slope of the latency increase varies depending on the chosen interpacket gap. Figure 3.12 clearly illustrates that the additional latency scales with the interpacket gap. Through our analysis, we have found that an interpacket gap equal to or larger than 32 bits effectively prevents any additional load-dependent latency.

Note that the increasing latency throughout the burst is only added inside the monitoring device. This measure shows the combined latency of the monitoring device plus the network. By changing the measure point to time stamping the packet when it is leaving the FPGA, we can measure the travel time of the packet without

considering the behavior of the monitoring device. This allows us to measure only the network performance, though the machines exchanging those packets will see the rising latency throughout the burst.

Hence, we recommend that servers generating bursts should utilize a larger interpacket gap to mitigate this effect. A 32-bit interpacket gap, for instance, can effectively nullify this latency increase during bursts.

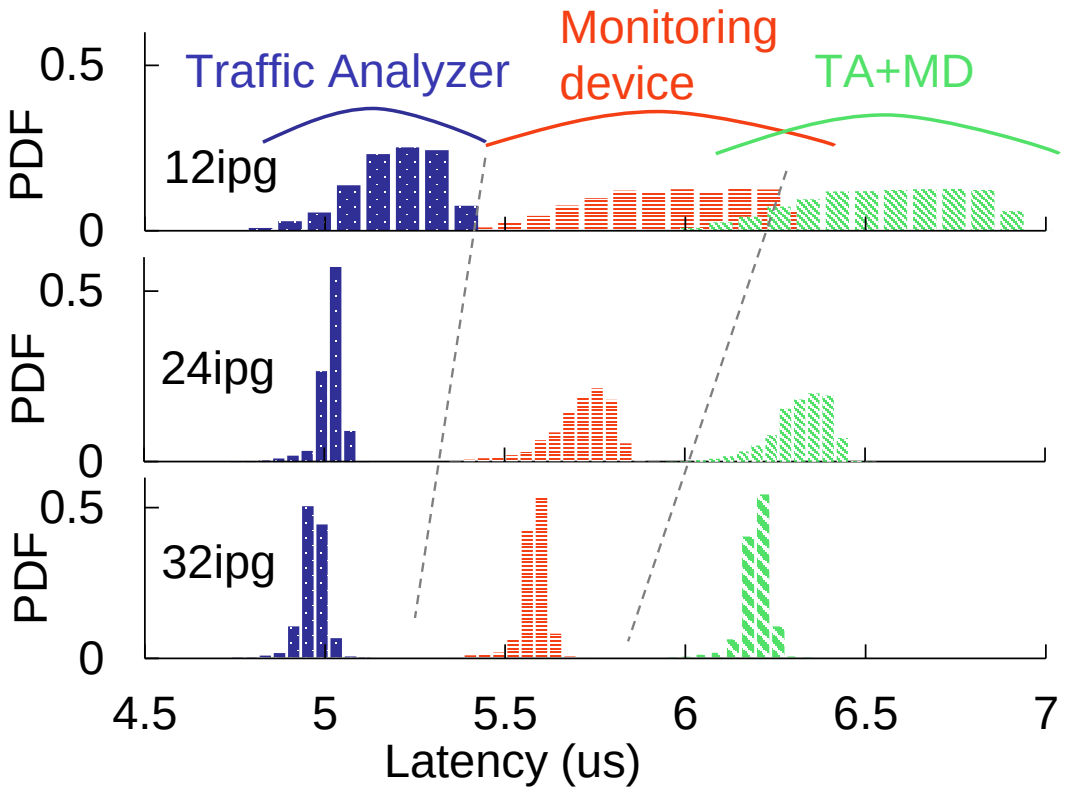

Figure 3.13: Latency distribution of 50 packets bursts for 12, 24 and 32 bits of minimum interpacket gap

Figure 3.13 demonstrates that the monitoring device is not the sole contributor to latency increase during bursts. We provide the Probability Density Function (PDF) of latency for a low-jitter network without flow competition, as measured by various devices. Initially, we measure the latency of bursts of 50 packets using the traffic analyzer without the monitoring device within the network. Then, we measure the latency using the monitoring device, and finally, we measure the latency with the monitoring device in the network path using the traffic analyzer.

Once more, we observe the latency shift caused by the processing time that we have not accounted for. Additionally, we notice that with low interpacket gaps (12 and 24 bits), the standard deviation of the latency distribution increases, while for the 32-bit interpacket gap, it remains relatively constant. This further underscores

that interpacket gaps shorter than 32 bits can have an impact on burst propagation or higher loads through the network.

When examining the first column, where the experiment is conducted with only the traffic analyzer in the network, we observe that the network struggled to handle the burst of traffic, leading to increased latency variation. Therefore, for time-critical applications that involve sending bursts of packets, it may be advisable to leave more bits between packets to avoid overloading the networking equipment along the communication path.

Impacts on throughput of the monitoring device

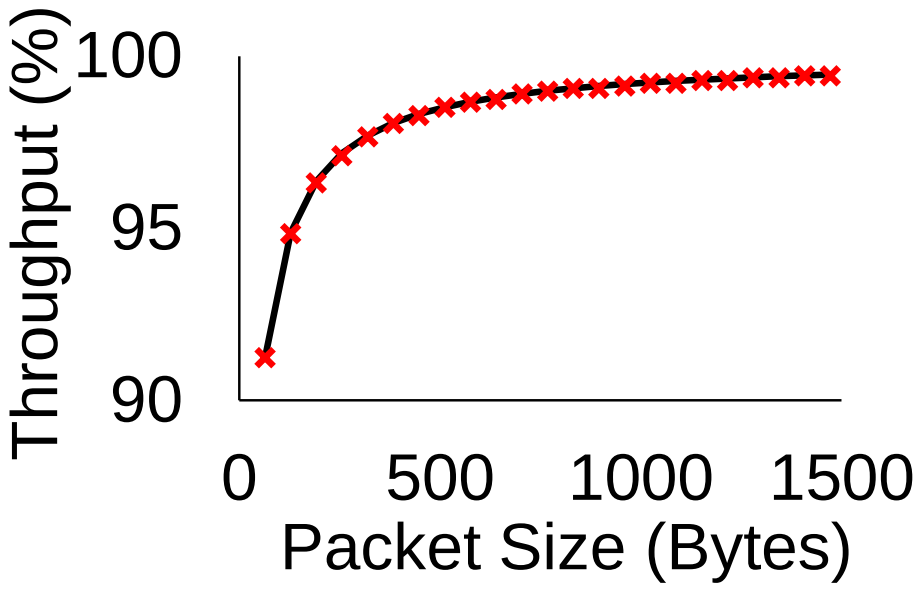

Figure 3.14: Throughput for constant bit rate traffic of different size.

In this project, we insert the timestamp between fields of the packet, thereby increasing its length. This increase in packet length can result in a reduction of throughput as observed from the perspective of the traffic source. In Figure 3.14, we examine how throughput evolves with varying packet sizes. For these measurements, we transmit constant bit rate traffic through our network. Each data point represents a flow of packets with the same length, and we gradually increase the traffic rate until we observe the first loss, indicating the maximum throughput for a packet of that length. To conduct this experiment, we begin with a packet length of 64 bytes, which is the minimum for Ethernet [31], and increment by 64 bytes up to 1518 bytes.

As in this experiment we monitor every packet, the overhead ratio depends on the packet length. Therefore, we can accurately predict the evolution of throughput with

varying packet lengths. In Figure 3.14, the red crosses represent the experimental results, while the black line represents our prediction. In the worst-case scenario, where we are monitoring only the smallest packets, the throughput is reduced to 91.25% of the line throughput. However, this scenario is only reached if all packets are of minimum size, and if all flows are monitored.

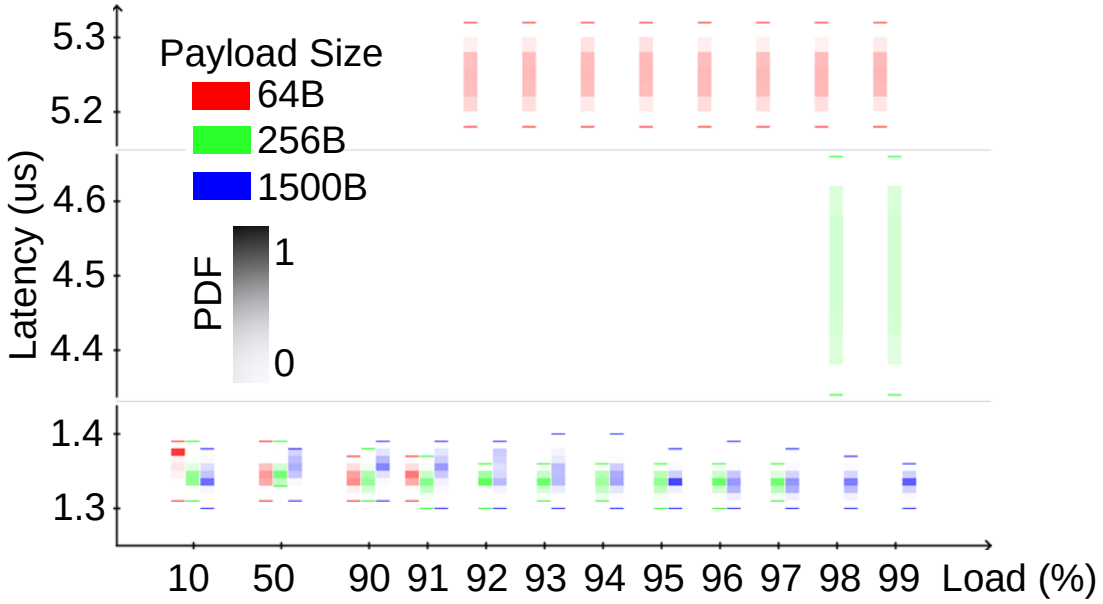

Figure 3.15: Latency distribution for constant bit rate traffic of different size.

In Figure 3.15 we report the latency as a function of the load and of the packet size while assuming constant bit-rate traffic. The load is varied by adjusting the interpacket gap. We can see the combined effect of the reduced throughput through the process of time stamping and the device behavior after the maximum throughput has been reached and loss starts to occur. We show the combined PDF for three packet sizes over different load. A bold line is placed to indicate the minimum and maximum of each distribution.

When the device is not under maximum strain, we measure we only represent two cases as the behavior does not vary much across different packet size and load between 10% and 90%. At these ranges the device measure a maximum 80 ns jitter across all packet size.

Afterwards, measuring load at each percent, we start seeing a jump in measured latency starting at 92% for 64 bytes packets and 98% for 256 bytes packets. These happen after the maximum throughput has been reached and are in accordance with the measures and predictions of throughput seen in Figure 3.14. Packet loss occurs because at a high throughput the insertion of the timestamp interferes with the

communication of following packets, the buffer in the ingress monitoring device fills up until packets are dropped due to a lack of buffer space.

We observe an increased latency because of the increased time packets spend in the full ingress buffer. This added latency depends on the packet length as the buffer can contain more 64 bytes packets than 256 bytes packets. Meaning more timestamps to be added to the flow of packet thus an increased waiting time.

The monitoring device brings low overhead, in the worst case scenario with only the smallest packet to monitor, it lowers the throughput by 9%. On average though, as not all applications will be monitored and packets will be longer, less than 1% overhead can be expected.

3.6 Conclusion

In this chapter we proposed a distributed monitoring plane for time-critical traffic. We developed it to be modular, making it adaptable for integration into various Ethernet packet networks. In this implementation, only latency and jitter, derived from latency are taken. This device enables continuous and precise latency monitoring in real time networks. Note that not all flows have to be measured. Through a flow identification mechanism, the user could choose which specific (time sensitive) applications they would like to monitor.

To achieve precise measurements across a network of distributed monitoring devices, we have established a synchronization protocol. These devices are organized in a tree-like architecture, with the root device serving as the source for daisy-chaining synchronization to the rest of the devices. By combining both time and frequency synchronization, we have been able to achieve a remarkable precision of 6.4 ns over a single synchronization segment. And the precision scales linearly with the number of segments. Frequency synchronization plays a critical role in maintaining the stability of time synchronization by mitigating drift that may occur in the local clocks of these devices. It is important to note that the resolution of the timestamps exchanged for time synchronization imposes an upper limit on the maximum precision achievable in the synchronization process.

Network operators have the flexibility to monitor specific applications by deploying the monitoring device within the network path and providing an application identifier. This identifier typically includes source and destination MAC addresses of the application. Once configured, the monitoring device will consistently timestamp every packet related to the specified application, enabling continuous latency monitoring to observe network behavior trends. At the egress monitoring device, timestamps

are extracted from the packets, and measurements are conveyed back through the monitoring network to an orchestrator for analysis and further action.

The introduction of monitoring devices into the communication path results in an increased overall communication time, adding approximately 1.2 µs of latency. This added latency is attributed to several factors, including the processing time within the FPGA, the deserialization and serialization of data, as well as the insertion and extraction of timestamps.

Under typical network conditions, the monitoring operations introduce relatively consistent latency with minimal jitter, typically staying under 20 ns. However, during moments of network strain, such as bursts of traffic or when the traffic rate approaches the maximum line throughput, the monitoring devices can introduce increased latency variation. We found that maintaining an interpacket gap greater than 32 bits, either through burst configuration or by limiting the maximum throughput, can mitigate these effects and allow the monitoring device to produce non traffic dependent latency increases with precise measurements.

Chapter 4

Clock tracking for improved latency stability

Contents

In this chapter, we focus on improving the method developed in Chapter 2. With our monitoring device, we observe the behavior of the latency evolution and its shortcomings and propose a different method for providing constant latency.

4.1 Shortcomings of distant clock frequency matching

In this section, we measure the latency evolution, to further understand the behavior of our system. We show the potential issue of the monitored behavior.

In the first chapter we demonstrated 50 ns Root Mean Square (RMS) jitter, as measured by the traffic analyzer, but the histograms showed more than 200 ns of peak-to-peak jitter. For the 5G use case, the peak-to-peak jitter needs to be contained under 130 ns which is not provided by our system.

With this discrepancy between RMS jitter and peak-to-peak jitter, we decided to investigate the behavior of the latency. Thanks to our monitoring device we measured the evolution of the latency over time of packets going through our DDX network.

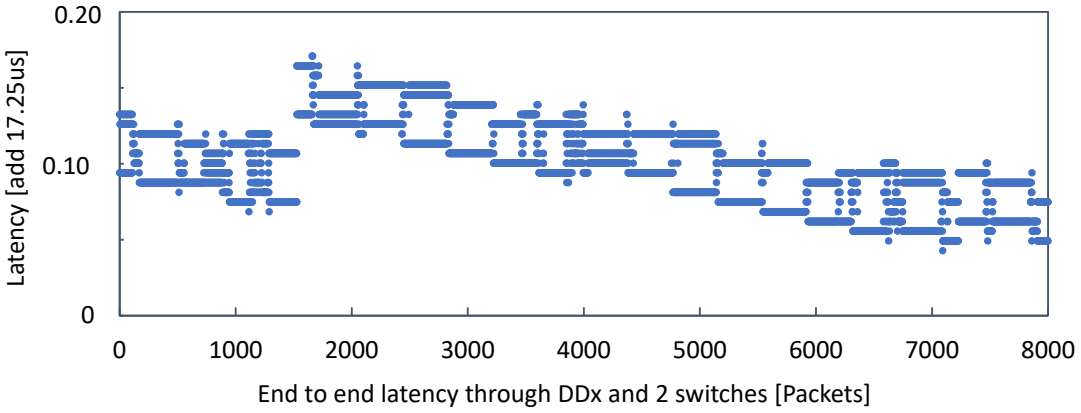

Figure 4.1: Latency evolution of time compensated packets.

In Figure 4.1 we present the evolution of latency over time of packets going through the network. Each sample representing a measure. The packet flow is constant bit rate traffic of packet of 256 bytes with 1 Gb/s data rate. The represented evolution lasts for 100 ms.

We can see that the latency of packets is slowly decreasing until it experiences a step and is increased again. This phenomenon is akin to a stone skipping on the surface of water, here the latency is skipping over a minimal latency threshold.

We can explain the behavior with the way we implemented the jitter compensation mechanism.

In order to decrease jitter, we recreated the delay between packets. To recreate the delay, at ingress we count the number of idle frame in between packets and

send it alongside the packet. This count is performed at the ingress local frequency, approximately 156.25 MHz with a small margin of error. At the egress device, to recreate the delay between packets, we emit the same number of idle frames based on the count that was made in the ingress device. These idle frames are emitted at the egress local frequency, approximately 156.25 MHz.

A difference in frequency will result either if the egress local frequency is slower in a constantly increasing latency, slowly but consistently filling up the buffer space until packets are lost. Either, if the egress local frequency is faster, it will result in a decreasing frequency until the packet that needs to be sent is not ready.

In Chapter 2, to account for the frequency difference we proposed to estimate the frequency difference though a linear regression of the rate at which the timestamps inserted in the DDX frame diverge. With this estimation we corrected and inserted or deleted idle frames to prevent any divergence during the jitter compensation process.

Due to frequencies that can change over time and network jitter that decreases the precision of the measure, we proposed to over-estimate the frequency difference. Therefore, after correction the egress local clock performing the jitter compensation is running marginally faster that the ingress clock. In Figure 4.1, we represent the latency jitter compensated packets over 100 ms. Each step is a clock cycle of 6.4 ns. At a given time the precision of the jitter compensation mechanism is of 7 clock cycles. However, during the complete measurement period the latency decreases by 14 clock cycles and is subjected to a discontinuity. A 14 clock difference over a period of a 100 ms corresponds to a frequency difference of just under 0.9 ppm.

The resolution of the frequency difference measure is 0.1 ppm. We separated both measurement points by 64 ms, leading to a detection of a minimum of 1 clock cycle difference over this period which corresponds to a 0.1 ppm clock frequency difference. We used a 0.5 ppm safety margin to be sure that the system will never be in the regime where latency is constantly increasing.

We now explain the behavior of the regime we place ourselves in, where the egress clock frequency is faster than the ingress clock frequency. We describe the different components of the latency over the whole path of the packet.

The latency experienced by packets can be divided into three sections. Time spent in the ingress device, time spent in the network and time spent in the egress device, which can further be divided in three categories. In Figure 4.2 we describe those different sections and the different contributions. This is an explanatory figure, some parameters may be exaggerated.

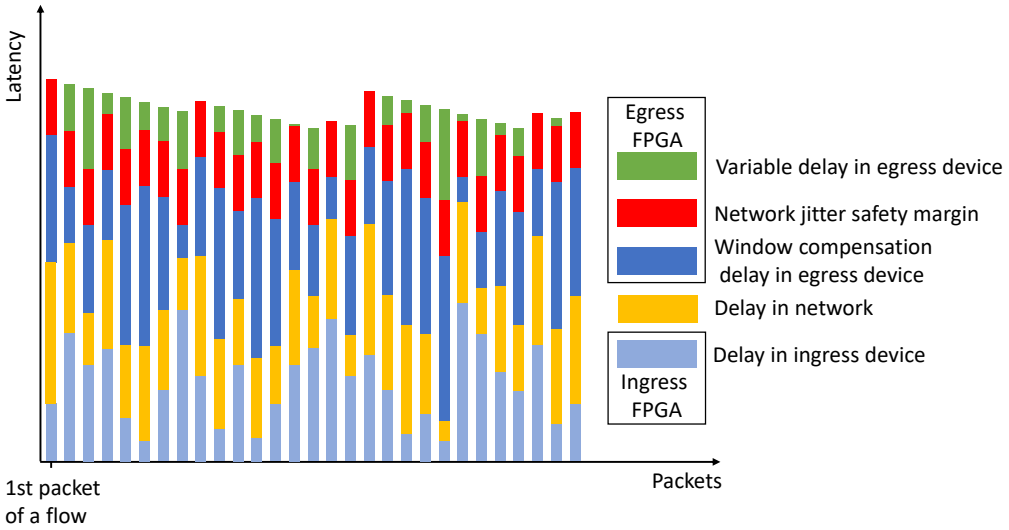

Figure 4.2: Latency contribution of different part in latency compensation

First, in the ingress device, represented in light blue in Figure 4.2, packets wait for their reserved slot to arrive. This waiting period is random based on the time of arrival and can last up to the period of the reservation window.

Second, when going through the network inside the DDX frames, the packet experiences jitter represented as the yellow bar in Figure 4.2. This latency is also random but capped thanks to the bridge setup between the ingress and egress device.

Third, the time spent in the egress device can be decomposed into three parts. Initially, the jitter added by the insertion in a periodic slot is compensated. This delay is represented in dark blue. For all the packet, adding the light and dark blue delay amount to the same waiting time, afterwards, only the network jitter needs to be compensated.

Then, in red in Figure 4.2, we have a waiting period caused by a safety margin to compensate for network jitter. We place this safety margin to reduce the probability of a frame needed to be sent be that has not arrived in the device due to a long network latency.

Lastly, in green we see the skipping stone effect that we detected in the behavior of our jitter compensation mechanism. The delay represented in green in 4.2 is the delay when the packet is ready to be sent, but the delay between packets has not been completed yet. This delay is variable and is mostly induced by the network jitter. Packets are ready to be sent once the correct number of idle symbols have been recreated. In our regime the egress clock, counting the number of idle frames at egress, is running faster than the ingress clock, so the delay between packets is slightly shorter than it was when entering the network. With 1 ppm frequency

difference if packets entered the ingress device with a 1 µs delay between them, at egress the delay is 1 ps shorter.

Over time this shortened delay between packets adds-up, and it appears as if the latency is decreasing. The latency decreasing until a packet is not ready to be sent. This is caused by high network jitter, symbolized by a tall yellow bar in Figure 4.2. This creates a skipping stone like behavior for the evolution of latency.

Because of the method we used for performing jitter compensation, we induced a behavior undesired for some applications. The RMS jitter we provide is low, but sometimes there is a discontinuity in the evolution of latency. Most often it is the difference of latency between consecutive packets that is most important. For example, running a synchronization application during one of these discontinuity would decrease the precision of the synchronization.

Moreover, this method of estimation of clock frequency difference is slow, 64 ms to set up and to detect change in frequency. It also assumes that the difference of frequency is constant over time creating a linear evolution of the offset between device over time.

In this chapter we introduce a new jitter compensation mechanism to prevent discontinuity in the latency evolution, providing lower peak-to-peak jitter. First we will thoroughly define the clock distribution problem.

4.2 Definition of the distant clock distribution model

In this section, we provide an analytical model for the distant clock distribution problem.

First, to distribute a clock signal we must define what a clock is. In this manuscript, we refer as clock the process of counting the passage of time. In real life we count time starting from two thousand and twenty-three years ago and divided time into hours, minutes and seconds. In computer systems the origin of time is often the 1 of January 1900, and a counter counts the number of nanoseconds elapsed since then. In computer time takes the form of a counter, an ideal counter C_{ideal} would track time exactly.

$$C_{ideal} = t$$

Because the time information is stored in bits, the counter has a maximum resolution based on the memory space. We also need to increment the counter to

provide a measurement of time. Most often the period of the increment gives the resolution of the clock and its maximum precision. We can use two clocks with a known fixed difference in frequency to create a system akin to a vernier providing better precision of time.

The clock thereafter becomes the number of time n the counter has been incremented at the frequency F_{REF}, and time become discretized.

$$C = n$$

For the experiments using a FPGA implementation, both the monitoring device and the DDX node, the reference frequency is $F_{REF} = 156.25$ MHz. This frequency is the most often used for controlling the logic of the 10Gbit Ethernet modules of the FPGA. We chose to use the same frequency for the user logic and the transceivers.

Although I programmed the clock generators on the FPGA development boards to produce a frequency of 156.25 MHz, a small imperfection in the quartz providing the base frequency can produce a clock signal that is not accurate. With the data sheet of the components, we can safely assume that the frequency difference is less than a 100 ppm.

In my implementation to carry the timing information across devices, I used 20 bits of memory. With the frequency of 156.25 MHz, giving the clock a resolution of 6.4 ns, the counter loop backs after iterating throughout the whole 20 bits in just over 6 ms ($2^{20} * 6.4$ ns). Meaning we have a precise, 6.4 ns, knowledge of time modulo 6 ms. Here the clocks do not need to be synchronized so there we don't have an ambiguity problem. In a further section we will discuss the minimum number of bits to be used to carry timing information in our prototype.

The clock being free running, unsynchronized to a master clock possess an offset θ to the ideal time.

$$C = n + \theta$$

In our problem we are trying to transfer timing information encoded on one clock frequency to another without having direct connection in between. We have two clocks, one in the ingress DDX add-on card C_i and one in the egress DDX add-on card C_e. We can write their formula with their respective frequency and offset.

$$C_i = n_i \frac{F_i}{F_{REF}} + \theta_i$$

$$C_e = n_e \frac{F_e}{F_{REF}} + \theta_e$$

As the information is encoded at the ingress frequency we will take $F_i = F_{REF}$.

In our previous approach described in Chapter 2 we tried to evaluate the parameter d to ensure that the same number of clocks cycle counted at the ingress or egress frequency would yield the same duration. This way $C_i - C_e$ is constant over time.

In this chapter, we try to keep the time difference between clocks constant not by estimating the drift but by making the egress offset θ_e evolve over time to compensate the drift.

4.3 Remote clock tracking

In this section, we present the different element for tracking a remote clock allowing us to recreate the traffic delays at the network egress.

With each slot we send timing information. The slot being periodical, each S clock cycles, in the implementation $S = 192$[1] clock cycles, we send the value of the ingress clock C_i. In the egress node, we store the value of the clock when the frame arrives C_e.

In an ideal network, with no jitter and a latency L we have:

$$C_i(0) - C_e(0) = L + \theta_i - \theta_e$$

If the ingress and egress frequency are equal then the clock difference will remain constant over time. Else the difference will vary. After n samples, the clock difference is:

$$C_i(n) - C_e(n) = L + n \cdot S \cdot d + \theta_i - \theta_e$$

We can define the drift d, the fraction of cycle created by the induced by the frequency difference during one clock cycle of frequency F_i.

$$d = 1 - \frac{F_e}{F_i}$$

If the ingress frequency is greater than the egress frequency, then $d > 0$ and the egress clock is running slower. Else $d < 0$ and the egress clock is running faster.

[1] It takes 190 clock cycles to send the DDX packet corresponding to the longest Ethernet packet (1518B) and we add 2 cycles for separation in order to avoid surcharge of the network.

Over time the clock difference will increase or decrease based on the sign of d. We now propose to follow the evolution of the term $n \cdot S \cdot d$ by operating on θ_e.

A complication is presented when the network communication is not perfect and latency can vary on the communication path. The clock difference becomes:

$$C_i(n) - C_e(n) = L + X + n \cdot S \cdot d + \theta_i - \theta_e$$

Where X is a random variable for representing the jitter. Based on the type of link X can follow different distributions. Latency often follows a long tail distribution, but can also tend to a normal distribution in the case of multiple preemption links or a uniform distribution for a single preemption link.

The problem is now to discriminate if the variation of the clock difference is caused by the network jitter or the clock drift. Network jitter is a high magnitude random process with no time correlation while the drift has low magnitude but offers a high time correlation. We therefore need to isolate the evolution caused by the clock drift away from the network jitter.

4.3.1 Clock tracking algorithm

In this subsection, we present the algorithm for performing remote clock tracking. The goal is keep the time offset between devices constant or within a predefined margin.

In Figure 4.3, we show the different stage of the clock tracking where the offset between clocks should be kept under 50 ns.

In blue, we show the information received before processing, we represent the evolution of the clock offset between ingress and egress device. On the link we have 100 ns jitter and a non-linear evolution of the offset over which further justify the need for a different approach for recreating the remote clock.

First we need to eliminate the jitter, we filter the input over many samples to create a clean signal represented in yellow in Figure 4.3. We will discuss the “filter” and the “many inputs” in a further section.

We then compare the filtered signal to the margin we chose and increase or decrease the local offset θ_e to track the evolution of the filtered signal. We represent in green the recreated signal. We can see that it evolve freely throughout the margin we used, ensuring that the clock offset is constant with a 50 ns precision. The constant offset over time between devices enables the recreation of inter packet delay with no diverging trajectory and no skipping stone effect.

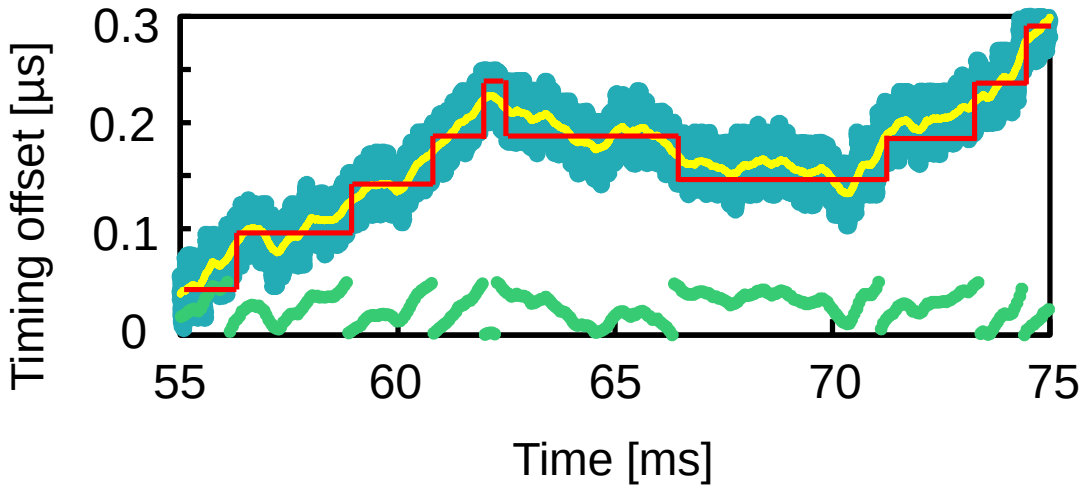

Figure 4.3: Remote clock reconstruction through clock tracking. Blue: Retrieved timing offset. Yellow: Smoothed retrieved timing offset. Red: Stored timing offset. Green: Corrected timing offset.

Here for the demonstration of the process to be more visual, we used a 50 ns margin. We can take this margin down to the resolution of the time stamp. In the following implementation will track the ingress clock down to the clock period of 6.4 ns.

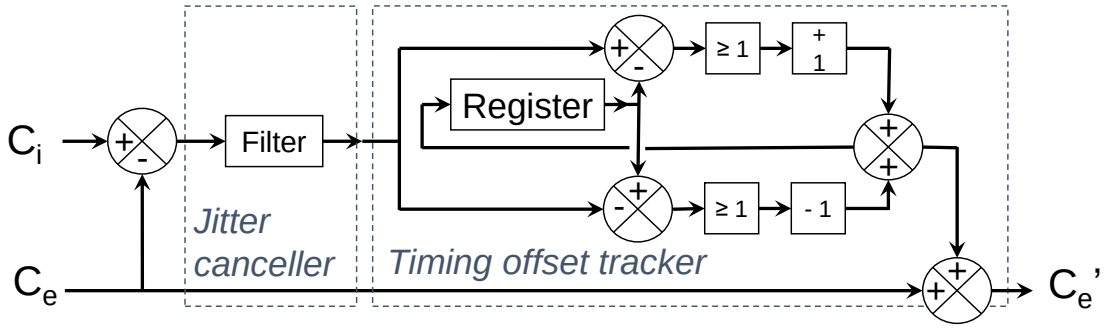

Figure 4.4: Remote clock tracking process.

In Figure 4.4 we present the implementation as performed on the FPGA of the module for tracking the ingress clock.

With DDX slot we receive a value of the ingress clock is given as the 20 bits word C_i. When receiving this timestamp, we store the value of the local clock as C_e. These are the two inputs of our module for tracking the ingress clock. The output of the module is a modified version of the local clock that track the ingress clock down to the clock period.

We first take the difference of those two values that produces a very jittered signal, which we filter. The goal of the filter is to cancel out the jitter and provide a clean signal only showing the evolution of the clock offset due to the drift between both devices. We will choose a filter based on how well it can reduce eliminate the jitter, and on how much lag it introduces. We will also consider the physical space the implementation of the filter takes in case memory space on a board is scarce.

The output value of the filter is updated with every new sample and compared to the previous one. If we detect a difference between the two we increment or decrement the egress offset to match the evolution of the output filter.

4.3.2 Filtering network jitter

In this subsection, we present the different filters we used for performing remote clock tracking through a network with latency variation. We analyze the lag they introduce and their stability.

As our goal is to cancel the network jitter which is a random process that we can associate with high frequency noise from the clock drift which produces a slow variation. We will therefore analyze low pass filters. In this section we study three filers:

- **An Infinite Impulse Response (IIR) filter**. The IIR filter does not require storing many past samples, for first order IIR filter a single value is stored, making it a great candidate for a low memory implementation.
- **An Moving Average (MA) filter**. The MA filter requires more memory is more efficient at cancelling white noise.
- **An Growing Average (GA) filter**. We propose a variation on the MA filter for improving the response time of the filter.

The Infinite Impulse Response filter

The IIR filter takes as input x_n, the sample constituted by the difference between the ingress and egress clock, and produces the output y_n, the filtered signal. The evolution of the first order low pass filter is given by this equation:

$$y_n = a \cdot x_n + (1 - a) y_{n-1} \tag{4.1}$$

In the equation 4.1, a is the weight the new sample is given compared to the previous output of the filter. In our case, as the network jitter is the major source of vari-

ation in an incoming sample and the overall trend is a slow evolution, we choose a to be small. We will simulate the behavior of the filter for $a = 2^{-8}, 2^{-9}, 2^{-10}, 2^{-11}, 2^{-12}$.

First we can analytically compute the behavior of our filter and its response to the clock drift. We compute the response for the case of zero network jitter. The response will vary according to multiple parameters:

- a the filter coefficient
- d the drift between the ingress and egress clock
- S the sampling period

The input x_n of the filter is the difference between the received timestamp from the ingress device and the local time at which this timestamp arrived. Assuming no jitter and a constant drift, the input can be formalized as such:

$$x_n = d \cdot S \cdot n + \Theta \tag{4.2}$$

The index n describes the number of the sample. The process of tracking the remote clock starts when the first DDX packet arrives, giving it the index 1. In Θ we hide all the constants, the latency of the communication between both endpoints, and the clock offsets which is none zero because at no point in time we are able to perform a time synchronization.

We first need to initialize the filter, finding a suitable y_0. We need a decent estimate of the clock offset. We reserve the first samples to compute an estimate. In my implementation, I gather the first 32 timestamps, compute their average and use it as y_0, giving $y_0 \approx \Theta$.

The maximum frequency difference allowed by the Ethernet norm is $d = 100$ ppm. A drift of one clock cycle is observed after 10 000 clock cycles (every 64us). With a sampling period of 200 clock cycles, the drift is observed at the 50th sample. Hence, choosing the initializing period to be 32 samples. Only the jitter will impact the initial value of the offset between the counters.

The response of a first order filter to a linear input as described by equation 4.1 is:

$$y_n = d \cdot S \cdot n - \frac{d \cdot S}{a}(1 - e^{-a \cdot n}) + \Theta \tag{4.3}$$

The first term, $d \cdot S \cdot n$, of equation 4.3 shows that the filter follows the input, and we can use its output to steer the local clock. The second term represents the

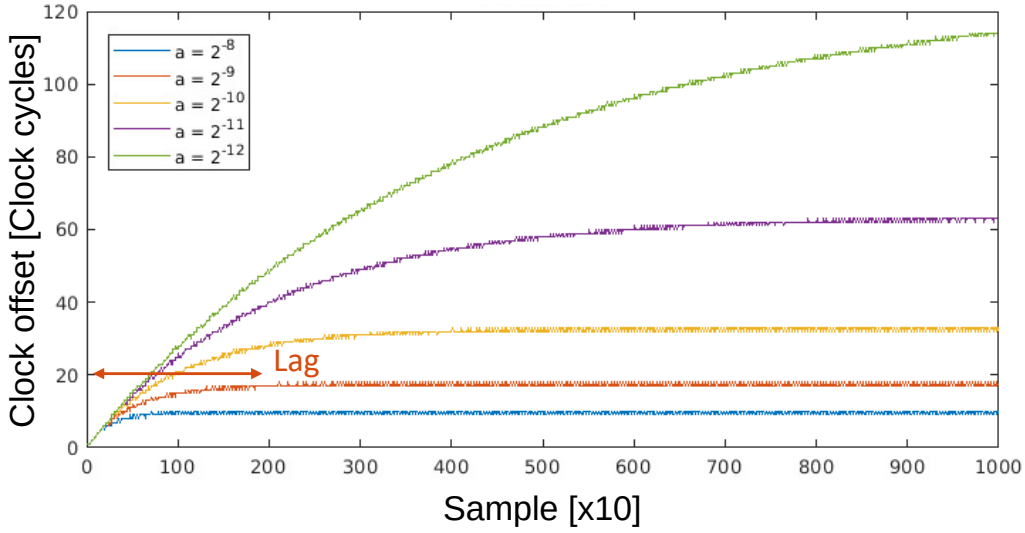

Figure 4.5: Simulation of the offset evolution boundaries for different parameter of Infinite Impulse Response (IIR) filters with no network jitter.

filter lag. We can use the lag term to compute the setup time of the filter. First we simulate the behavior of the filter.

We run a behavioral simulation of the whole remote clock tracking process. We used the output y_n of the filter to modify the egress local offset following the evolution of y_n. In Figure 4.5, we show the offset between the ingress clock C_i and the local modified egress clock C'_e.

In the simulation, during the setup time, we can see the exponential growth reaching an asymptote. The lag introduces an offset of magnitude $\frac{d \cdot S}{a}$. The offset results of the time taken by the filter to track the input. The offset is accentuated when the drift between clock increase and increase when the filter coefficient decreases.

To compute the lag we first need to define a rule to know whether a filter has completed its setup. The filter is properly setup when it follows the input with enough precision. We therefore define a condition on the finite difference of $y_n - x_n$ giving how many clock cycles can change over a second.

We can compute from which sample N the filter produces an evolution of less than B clock cycles per second.

$$N = \frac{-1}{a} \cdot \ln\left(\frac{B \cdot a}{F_e \cdot d(1 - \exp(-a))}\right) \tag{4.4}$$

The evolution of this term described by equation 4.4 is mostly linked to the filter coefficient, evolving almost linearly with the inverse of the IIR filter coefficient. For example the theoretical setup time of an IIR filter of parameter 2^{-8} is 5.9 ms. After this time, less than a clock slip will happen per second.

We can now look at the behavior of the filter when there is network jitter on the DDX bus. We consider jitter on the communication path of amplitude $2J$. We will consider that all the packets arrive with a latency comprised between L and $L + 2J$. We define $L' = L + J$ to represent the jitter by a zero mean random variable. We assume that the jitter of each packet is independent, and we compute the behavior of the filter for a normal distribution of the jitter.

The input of the filter described by equation 4.2 becomes:

$$x_n = d \cdot S \cdot n + X_n + \Theta \tag{4.5}$$

In Θ, L is replaced by L', the average latency. X_n, representing the jitter, is a random variable following a normal distribution of zero mean and of standard deviation $\frac{J}{3}$ so that 99.7% of packets have a latency comprised between $L' - J$ and $L' + J$.

$$X_n \sim \mathcal{N}(0, (\frac{J}{3})^2), \forall n \in \mathbb{N}^+$$

In the filter design we choose a to be small to limit the impact of a new sample on the previous step of the filter. After filtering a sample, the ouput follows the distribution described by equation 4.6.

$$Y_{n+1} \sim \mathcal{N}(0, (1-a) \cdot \sigma_{Y_n}^2 + a^2(\frac{J}{3})^2) \tag{4.6}$$

The standard deviation is the sum of two terms, one due to the uncertainty of the previous estimation, and the second to the uncertainty of the current sample.

In Figure 4.6, we simulate the behavior of the filter when the input contains 100 clock cycle of uniformly distributed jitter for various filter coefficients. Here we can see the importance of choosing the filter coefficient. With a smaller coefficient, the set-up time is shorter, but the output is less stable. And the stability depends on the amount of jitter inserted by the network.

Based on the quality of service to be provided and the communication path we will be able to appropriately choose the coefficient to minimize the lag whilst ensuring enough clock stability to provide constant latency.

We can compute the behavioral evolution of the filter considering that all the samples have independent jitter of the same magnitude. With the updated definition of x_n in equation 4.5 we have:

$$y_n = d \cdot S \cdot n - \frac{d \cdot S}{a}(1 - e^{-a \cdot n}) + Y_n + \Theta$$

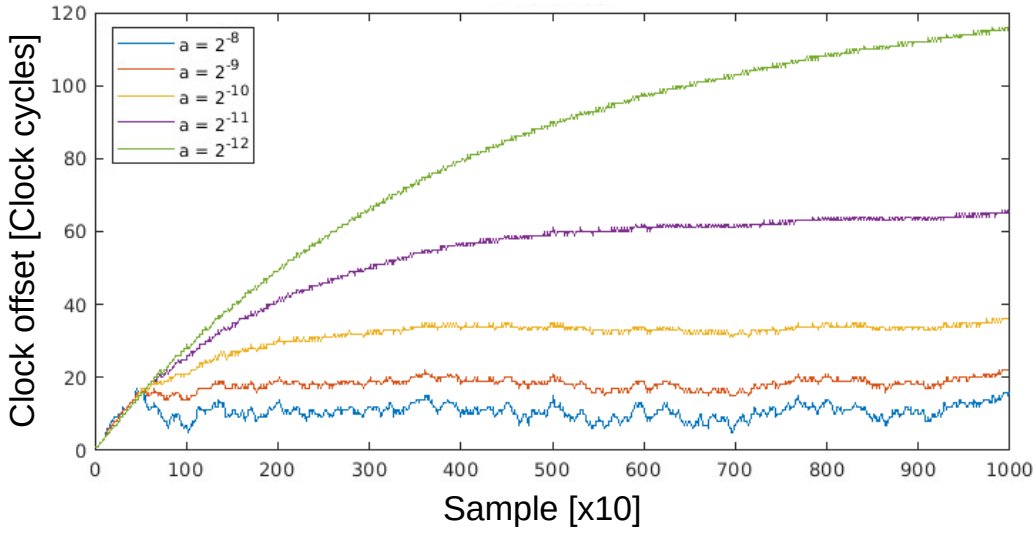

Figure 4.6: Simulation of the offset evolution boundaries for Infinite Impulse Response (IIR) filters.

Where Y follows a normal distribution:

$$Y_n \sim \mathcal{N}(0, (\frac{J}{3})^2 a^2 \frac{(1-a)^{2n}-1}{(1-a)^2-1})$$

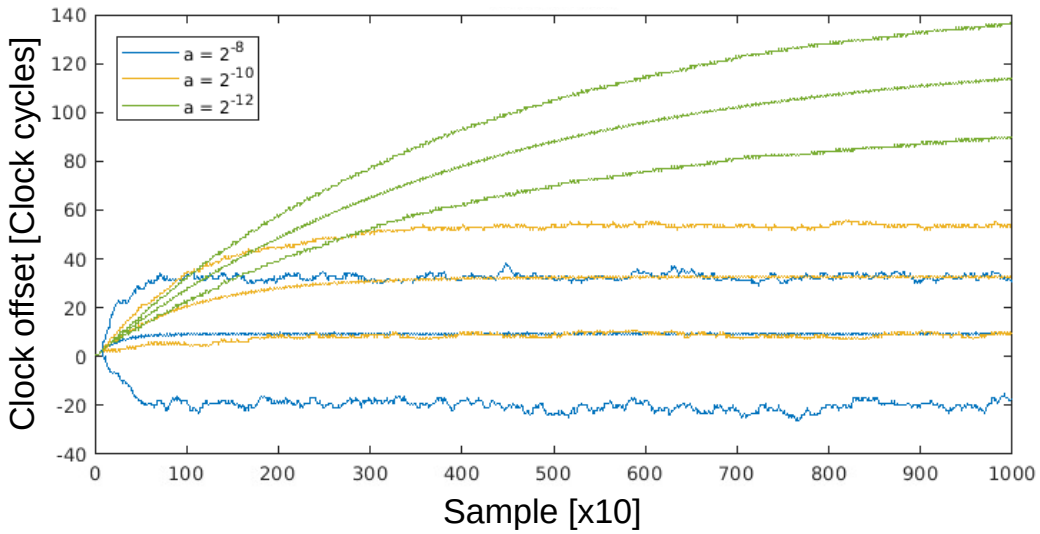

Figure 4.7: Simulation of the offset evolution boundaries for Infinite Impulse Response (IIR) filters.

In Figure 4.7, we run the previous experiment, with 100 clock cycle of uniformly distributed jitter, a thousand times and represent the minimum, average and maximum offset for each incoming sample. The average curve follows the same evolution as the one in Figure 4.5 which shows the zero mean aspect of the random distribution.

Because of jitter, the offset between clocks can vary, mostly during the setup phase. Though a maximum wander can be defined as the number of sample tends toward infinity, the offset can vary up to $\frac{J}{3}\sqrt{\frac{a}{2}}$ clock cycles. Though we can not predict the final state of the clock offset, it is not an issue as we only want a constant offset over time, and we saw the stability of the filter.

The Moving Average filter

Often time, in signal processing a moving average filter is used to smooth out an input. In our case we want to eliminate the noise jitter makes on the slow evolution of the clock offset. We will therefore use a moving average filter to track the evolution of the offset. With each new sample we compute the average of the past M samples:

$$y_n = \frac{1}{M} \sum_{i=n-M+1}^{n} x_i$$

With each new samples we can compute the sum of these M terms, or we can rewrite the formula to use the past output of the filter.

$$y_n = y_{n-1} + \frac{x_n - x_{n-M}}{M}$$

For the M first terms the average is undefined. We perform the same initialization as with the IIR filter, with the first samples we compute an average, which gives use an estimation of Θ, and use the result for the missing terms. In my implementation I use an average of 32 terms which takes about 40 µs to compute with samples coming in every 192 clock periods. We therefore consider the offset between counters fixed before the start of the process:

$$x_n = \Theta_0, \forall n \leq 0$$

First, we compute the output of the filter with no input jitter. We use the same input as the IIR filter, described by equation 4.2, we have:

$$y_n = \begin{cases} d \cdot S\frac{n(n+1)}{2M} + \Theta_0, n \leq M \\ d \cdot S(n - \frac{M-1}{2}) + \Theta_0, n > M \end{cases} \tag{4.7}$$

Equation 4.7 shows two distinct behaviors. During the setup we are always subtracting the same value, only the new sample contains information about the slope of the drift between clocks. If we wait until having enough samples to perform the filtering, the clock offset would linearly increase instead of being dampened by the second degree polynomial evolution.

When $n > M$, we have two terms, the first $d \cdot S \cdot n$ shows that the output follows the evolution of the input, and the second $d \cdot S\frac{M-1}{2}$ is a constant offset that was created during the filter setup.

We can easily define the filter lag which is to the time it takes to fill the buffer. In a same manner as for the IIR filter, we have a trade-off between the lag and the precision of the filter.

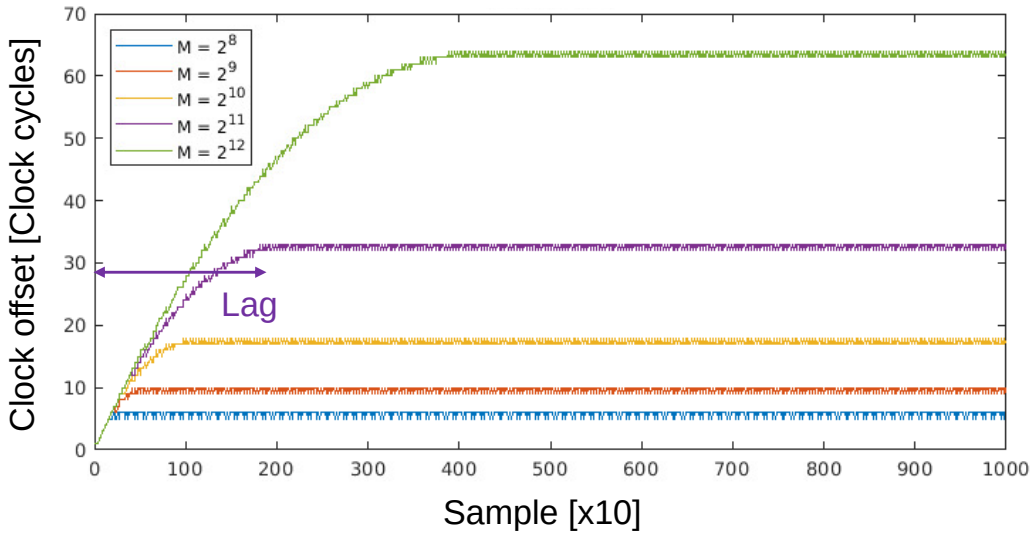

Figure 4.8: Simulation of the offset evolution boundaries for different parameter of Moving Average (MA) filters with no network jitter.

In Figure 4.8 we run a behavioral simulation of the MA filter for different amount of sample used to perform the average. M in MA filter plays a role similar to $1/a$ where when it increases the stability of the filter increases as well as the lag.

During the setup time the drift causes a clock offset given by $d \cdot S(M-1)/2$. As we do not initially know the value of the drift d, we can not predict the offset. Therefore, we must start the jitter compensation after the filter set-up.

We can study the behavior of the filter with network jitter. We consider the same input jitter, described by equation 4.5, with X_n the jitter term following a normal distribution of zero mean and a standard deviation of $J/3$.

To limit the impact of the jitter on the output of the filter we choose M to be big. After one step of the filter the next output of the filter follows this distribution:

$$Y_{n+1} \sim \mathcal{N}(0, \sigma_{Y_n}^2 + \frac{2}{M^2}(\frac{J}{3})^2)$$

We saw that a and $1/M$ play a similar role in the filtering process. But because each computation of the MA filter involves two samples, for the same parameter,

$a = 1/M$, the MA filter output has a standard deviation greater by a factor of $\sqrt{2}$ than the IIR filter output.

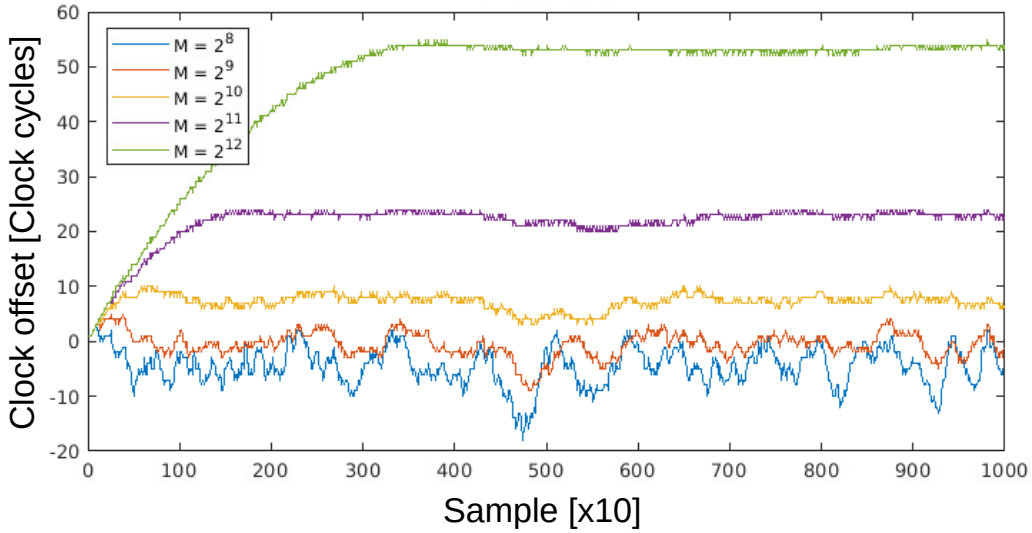

Figure 4.9: Simulation of the offset evolution boundaries for Moving Average (MA) filters.

In Figure 4.9, we show the behavior of the filter when the input contains 100 clock cycles of uniformly distributed jitter. For equivalent coefficient, we can see a greater evolution in this figure compared to the behavioral evolution of the IIR filter presented in Figure 4.6. The MA filter present less stability than the IIR filter for equivalent coefficient.

The drawback of lack of stability is greatly reduced by the faster set-up time. For equivalent coefficient, the set-up time of the MA filter is reduced by a factor superior to 4.

We can compute the overall behavior of the filter considering that all samples have independent jitter. We have:

$$y_n = \begin{cases} d \cdot S\frac{n(n+1)}{2M} + Y_n + \Theta_0, n \leq M \\ d \cdot S(n - \frac{M-1}{2}) + Y_n + \Theta_0, n > M \end{cases}$$

Where,

$$Y_n = \begin{cases} \frac{n}{M^2}(1 + \frac{1}{32})(\frac{J}{3})^2, n \leq M \\ \frac{1}{M}(1 + \frac{1}{32})(\frac{J}{3})^2, n > M \end{cases}$$

For the MA filter, the initial value impacts the distribution of the filter output as during the M first samples this same value, with its uncertainty, is subtracted.

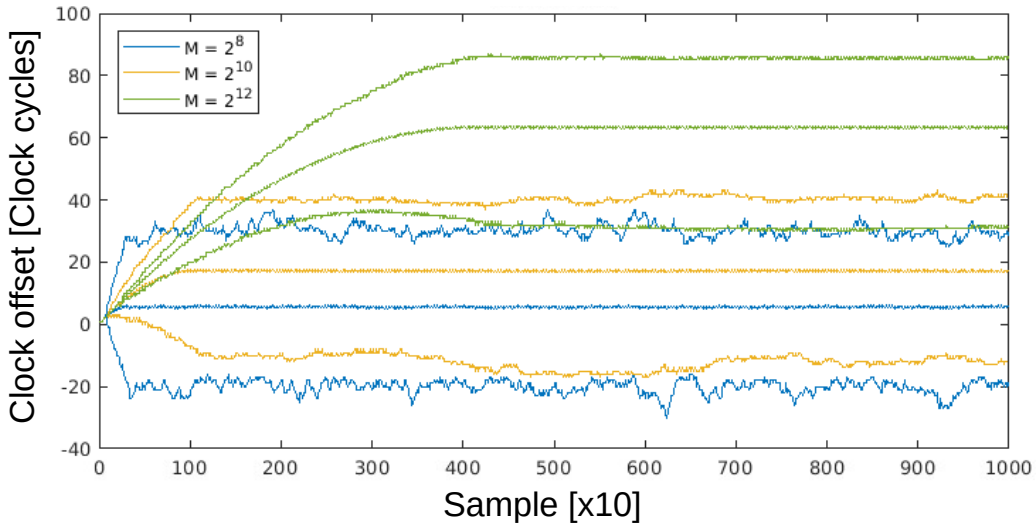

Figure 4.10: Simulation of the offset evolution boundaries for Moving Average (MA) filters.

In Figure 4.10, we run the previous experiment, with 100 clock cycles of uniformly distributed jitter, a thousand times and represent the minimum, average and maximum offset for each incoming sample. The difference between minimum and maximum is greater for the MA filter than for the IIR filter, but after the setup it stops increasing and the offset is stable.

The Growing Average filter

We propose the Growing Average (GA) filter, a modification on the MA filter based on the observation that after the setup, the output of the MA filter exactly follows the evolution of the input. We leverage the setup time of the filter with a lower averaging period to provide a degraded performance until having having a bigger period to average over.

We increase in steps the filter coefficient of a MA filter, passing to the next coefficient once enough samples are stored to not rely on the initialization value for computing the output. In the implementation we choose to successively use each powers of two until reaching an appropriate coefficient for dealing with the network jitter. The user can choose the final coefficient. In the following implementations, we stop increasing the coefficient once we have reached 2^{12} samples.

We still initialize the filter with the 32 first samples, so that the jitter of the first samples do not create a huge discontinuity in the clock offset. We define θ_0 as the average of the 32 first samples.

The filter can be defined by this equation:

$$y_{n+1} = y_n + \frac{1}{\min(2^{12}, 2^{\log_2(n)})}(x_{n+1} - x_{\min(2^{12}, 2^{\log_2(n)})})$$

With this filter, for the whole evolution, the filter behaves as an initialized version of the MA filter of coefficient $\min(2^{12}, 2^{\log_2(n)})$

Because we are in an initialized version of a MA filter after the 32 first samples, the evolution of the filter without jitter can be described as:

$$y_n = \begin{cases} \Theta_0, n \leq 32 \\ d \cdot S(n - 32d \cdot S) + \Theta_0, n > 32 \end{cases}$$

A small offset is inserted during the setup of this filter, and after it follows perfectly a linear input.

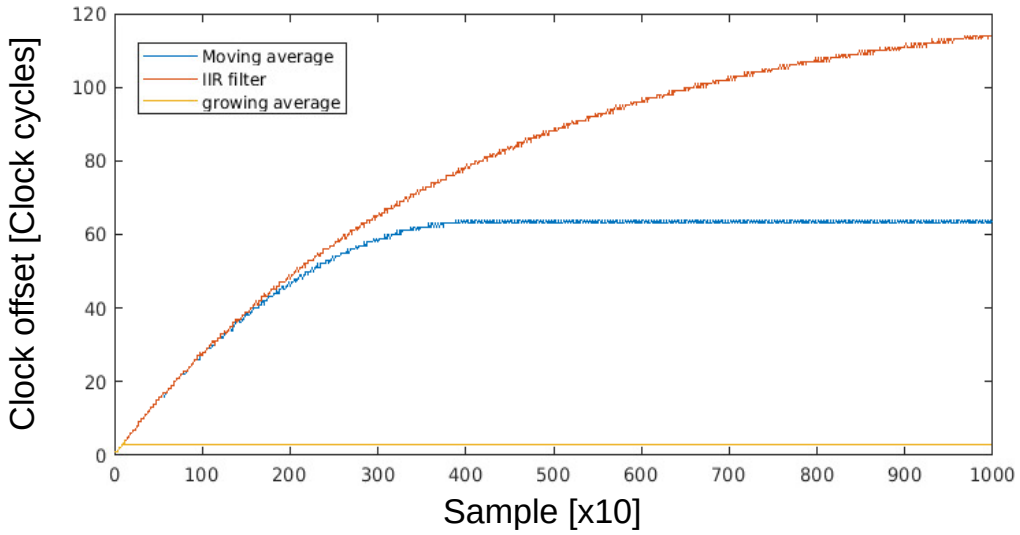

Figure 4.11: Simulation of the offset evolution boundaries for different filters with no network jitter.

In Figure 4.11, we compare the evolution of the GA filter to a MA filter of coefficient 2^{12} and an IIR filter of coefficient 2^{-12} without input jitter. We can see with the behavioral simulation that a minimal offset is imparted during the filter setup and afterwards the clock offset is corrected and is constant.

The drawback of using the GA filter is that during the first samples, until a high coefficient is used, the output of the filter is highly affected by the randomness of the input. Keeping the same jittered input as described by equation 4.5, the output distribution evolve as described by:

$$Y_{n+1} \sim \mathbb{N}(0, \sigma^2_{Y_n} + \frac{2}{\min(2^{12}, 2^{\log_2(n)})^2}(\frac{J}{3})^2)$$

During the first samples the output of the filter is highly affected by the jitter hence the process to track the clock will perform compensation based on the jitter, resulting in the creation of a large distribution of the clock offset.

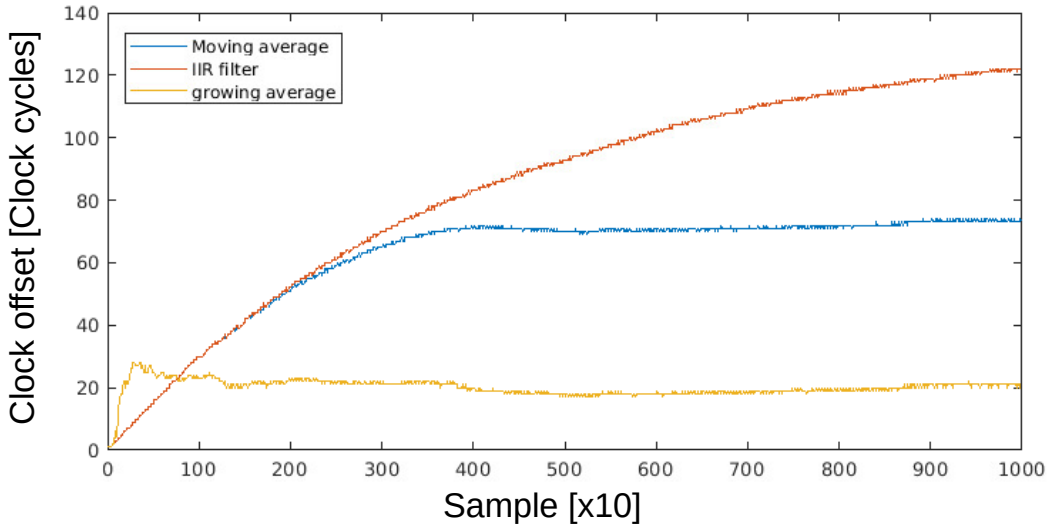

Figure 4.12: Simulation of the offset evolution for different filters.

In Figure 4.12, we compare the simulated evolution of the three filter with 100 clock cycles of uniformly distributed jitter. We observe a rapid evolution of the offset for the GA filter and a faster stabilization toward its end value.

The global evolution of the GA filter is highly skewed by the first samples, they set the offset that will be maintained over time. Whilst the filter coefficient is growing the standard deviation is subjected to small variation. At each power of 2, when the coefficient is increased we have:

$$Y_n \sim \mathbb{N}(0, \frac{1}{32}(\frac{J}{3})^2)$$

After completing the growth the standard deviation of offset distribution keeps this value. The compensated offset value cannot be predicted compared to the other filters, enough time must be given to the filter to stabilize.

In Figure 4.13, we compare the simulated evolution of the three filter with 100 clock cycles of uniformly distributed jitter. We represent the minimum, average and maximum offset for each incoming sample. The growing average filter is the only filter to produce an output centered around zero, regardless of the drift between clocks.

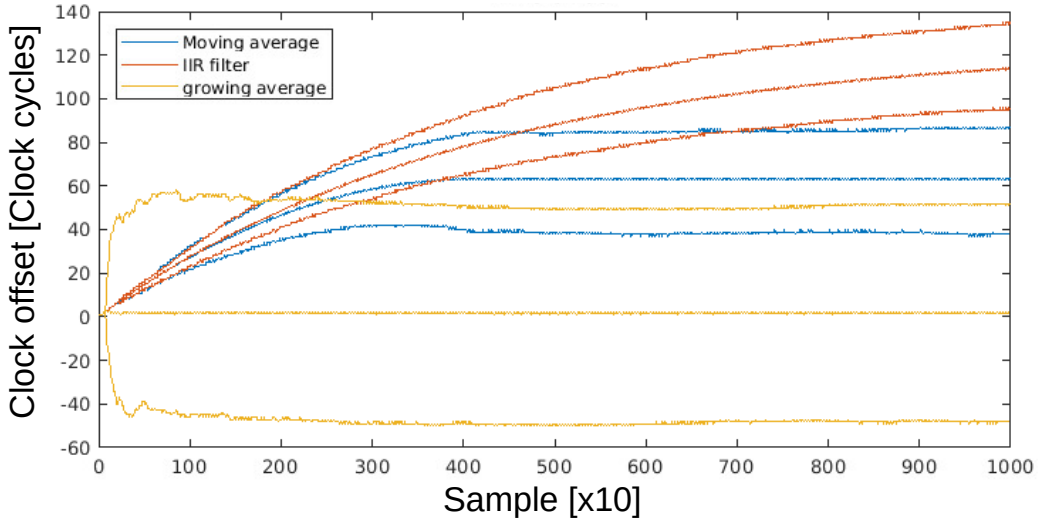

Figure 4.13: Simulation of the offset evolution boundaries for different filters.

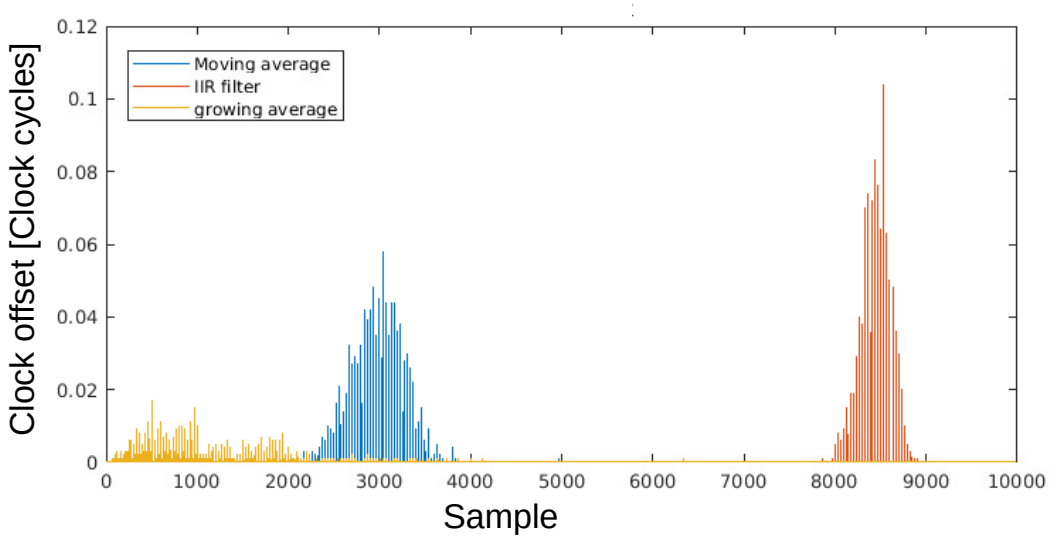

Figure 4.14: Probability Density Function (PDF) of time to lock for various filter.

In Figure 4.14, we ran a thousand simulation for each filter and saved the last sample where the rest of the evolution is less than 3 clock cycle away from the offset at the end of the simulation. This gives criteria to describe the stability of the filter, which we call time to lock. We display the PDF of the time to lock for the 3 filters.

We can see that with a high consistency the MA filter is stable around a thousand samples before the end of its setup of 4096 samples. The slope of the input is almost preserved at the end of the setup.

The IIR filter does not fully have the time to complete the setup. As the end value, we compare the offset to, is the value reached after 10 000 samples, the slow

evolution of the filter allows to be within three clock cycles of the end value within 8500 samples.

As the evolution of the GA filter is the most subjected to network jitter, its distribution is more widely spread. A great use case for this filter is when the application does not have the stringiest latency requirements. This filter allows to serve the application faster while ensuring a higher stability than required after a long period.

4.3.3 Timestamp minimal size

In this section we compute the minimal size of the timestamp as a function of the relative clock drift (d) between transmitter and receiver clocks, of the sampling period (S), of the filter and the filter coefficient (a), and of jitter (J).

In our process for tracking the clock evolution, we send periodic timestamps. In my simulation and implementation, I use a 20 bit word to send the timestamp, mostly because I use a 20 bit counter to count time locally. Actually a few bits could be saved thus increasing by a small margin the useable bandwidth of the DDX bus.

To find how many bits are necessary to carry time information between remote devices, we need to re-understand the clock tracking mechanism. Two remote devices have their local time, increasing at the frequency of their local oscillator. As the oscillators are not perfect, they do not beat at the same frequency. The 10G Ethernet norm [31] guarantee that the frequency difference between devices is below 100 ppm. With the maximum frequency difference, the offset between the two counters in the remote devices is increased by one every 10 000 clock cycles.

If we could directly compare the parity of the remote counters, comparing the last bit of the counter, we could see the phase shift process happen every 10 000 clock cycle. Directly comparing the counters is equivalent to sending the time information with no network jitter.

If the network jitter is null, only one bit is required to carry timing information to perform clock tracking for the jitter compensation mechanism.

Now, if the jitter is non-null. In the hardware implementation we use a fixed length binary counter of n bits. The counter counts from 0 to $2^n - 1$, and the next increment brings back the counter value to 0. This is called a counter overflow and in any implementation we must deal with this issue. Or we could use a 64 bit counter that would overflow in 3743 years.

With the remote clock tracking mechanism we try to keep the difference of value from the counter constant. The input from the mechanism is:

$$C_i - C_e \equiv \Theta + X \mod 2^n$$
$$\Theta = L + \theta_i - \theta_e$$

With X a random variable representing jitter comprised between J and $-J$. In a short period of time the mechanism input will receive values in the range $[\Theta - J; \Theta + J]$ mod 2^n. We need to compare all these value even during a counter overflow. In this case we will need to implement that a value of 1 is greater than a value of $2^n - 1$ and that their average is 0 and not 2^{n-1}.

To perform the filtering operation all the samples to be unambiguously described by the counter, meaning that $2J < 2^n$. If such condition is true then no two value with a high jitter difference can have the same counter value. If this condition is not respected the output of the filter is skewed, and we are not able to take commendable action from its output.

4.4 Constant latency enabled by remote clock tracking

In this section, we implement our method to track a remote clock enabling a faithful recreation of the delay between packets. We experimentally demonstrate our prototype over a network with emulated jitter. We purposefully desynchronize our FPGA to test the robustness of our protocol.

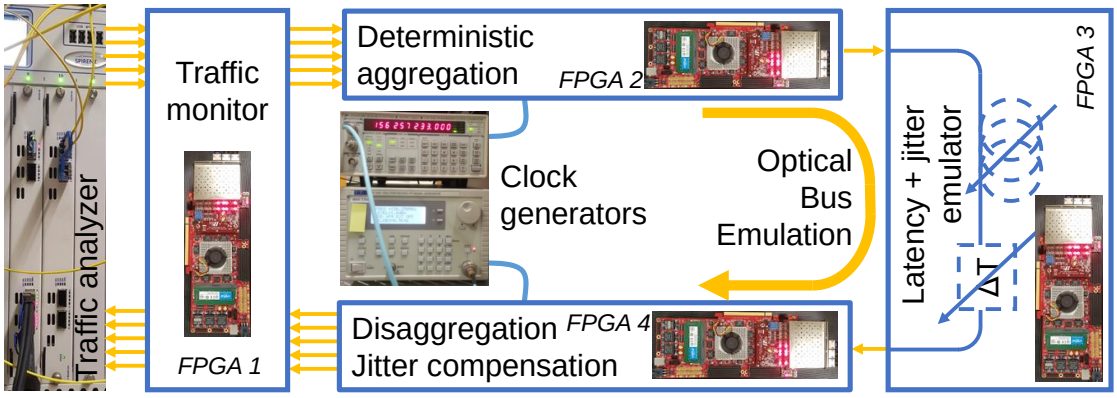

Figure 4.15: Experimental setup of deterministic optical bus.

In Figure 4.15, we describe the experimental setup for monitoring the latency through the DDX bus. With a Spirent traffic generator/analyzer we generate a flow of packets, to be inserted in the DDX bus. First the packets enter the traffic

monitoring device we implemented in a FPGA so that we are able to measure the latency evolution. For these experiments, we added a feature in the monitoring device to measure the latency distribution. This feature is already present in the Spirent, but we only have access to 16 configurable bins of latency with the minimal precision of 10 ns. In our device, I implemented 512 bins of each a clock cycle width so 6.4 ns.

After the traffic has been timestamped by the monitoring device, we send it to the ingress DDX add-on card where it is inserted in its dedicated time slot and sent to the network. To study the jitter compensation mechanism, I have implemented a latency and jitter simulator. While we can send the traffic through a network of switch and insert competing traffic to create jitter, we are not able to precisely control the amount of jitter on the communication path. So we chose to implement our own device to control the amount of jitter.

After the latency and jitter emulator we traffic is sent to the egress DDX add-on card where we perform jitter compensation. To test the reliability of our method we have purposefully created a clock drift of 50 ppm between the ingress and egress device. We used two clock generators to drive the local oscillator of each FPGA with the desired frequency difference.

Afterwards the traffic is sent back to the monitoring device and the traffic analyzer where we can collect latency monitoring data. We will be able to measure the peak-to-peak jitter and the RMS jitter. Whilst the measurements in Chapter 2 are only of RMS jitter.

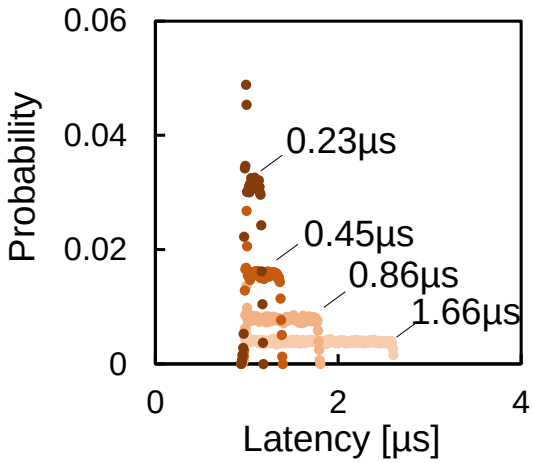

Figure 4.16: Measured distribution of emulated latencies (jitter).

In Figure 4.16, we show the distribution of the latency provided by our latency and jitter emulator. I implemented the emulator to create a uniform distribution

with a predefined width. We use hardware random number generator to give a random latency for each packet inside the emulator.

4.4.1 Providing constant latency with the moving average filter

In this subsection we show the behavior and performance of the Moving Average (MA) filter.

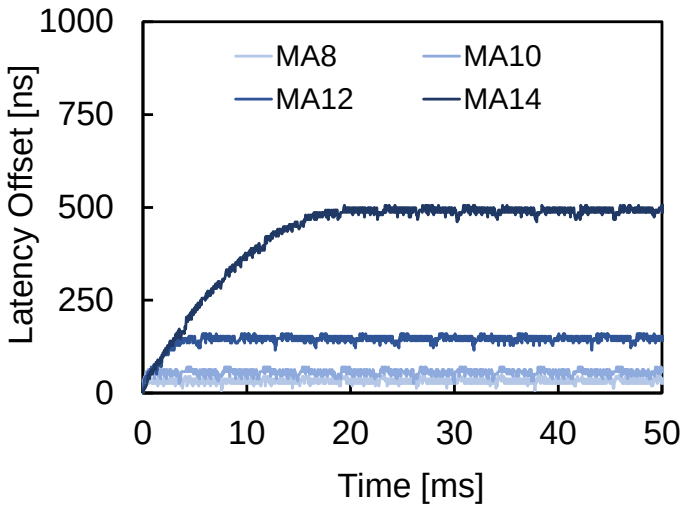

Figure 4.17: Moving average (MA) filter response for following 50 ppm clock drift with no network jitter.

In Figure 4.17, we show the latency offset created at the beginning of the communication on a path with no jitter. We start the communication of DDX packets as soon as packets arrive in the client ports. By measuring the latency of client packets we are able to see the evolution of latency during the bus setup. We choose the zero offset to be the latency of the first packet.

We see the same evolution as described in equation 4.7. This experiment demonstrates the rapid setup time of the MA filter. The higher the coefficient is the slower it takes to set up, for the moving average of 2^{14} samples, the filter takes about 20 ms to follow the input. Whilst it takes 4 ms for 2^{12} samples, and less than 2 ms for the other two coefficients.

We can see the offset created by the high, 50 ppm, frequency difference, and we see the same the evolution that we analytically computed. Compared to the previous method of jitter compensation described in Chapter 2, we do not see the skipping stone effect and a latency evolution which is continuous.

In the evolution, for every filter coefficient we see periodical drops in latency, of maximum five clock cycles. Because we induced a frequency difference between the ingress and egress card of 50 ppm, a frequency difference within the same range is probably occurring with the monitoring device. Due to the frequency difference between communicating devices, few clock periods need to be lost or caught back in the transmission module of the FPGA. In the implementation, I used the 10G PCS/PMA module by Xilinx which has an elastic buffer to deal with frequency difference. So the packets might not experience the exact same latency in the point to point connection between FPGA creating a periodic drop of latency. We could observe the same phenomenon in Figure 4.1 with the periodic change of latency.

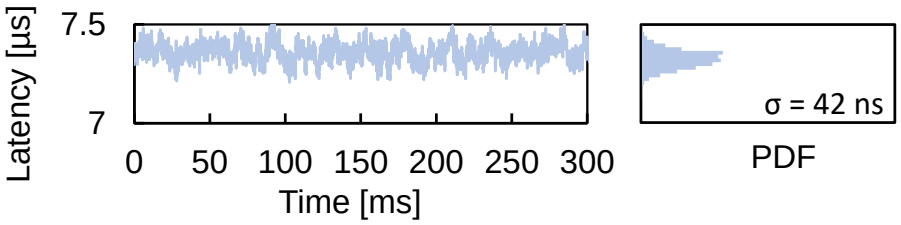

Figure 4.18: Latency evolution (over 300 ms) and distribution (over 1 minute) after compensation using moving average (MA) filter of size 2^8 with 1.8 µs network jitter.

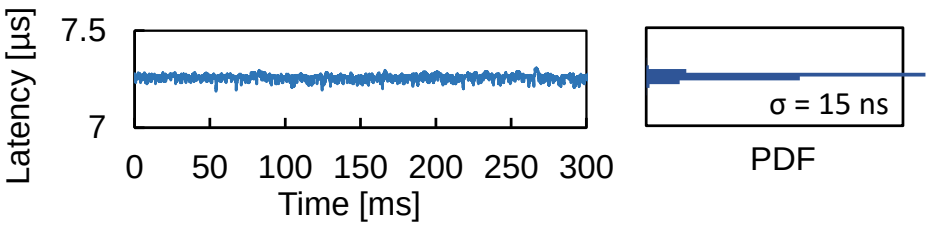

Figure 4.19: Latency evolution (over 300 ms) and distribution (over 1 minute) after compensation using moving average (MA) filter of size 2^{12} with 1.8 µs network jitter.

In Figure 4.18 and Figure 4.19, we present the latency evolution and distribution once the filter has completed its setup stage. It represents the working condition of the latency compensation, and show to which extent we are able to reduce the latency variation over a jittered network path.

In Figure 4.18, we show that we are able to reduce a network jitter of peak-to-peak 1.8 µs down to 211 ns peak-to-peak jitter. With only 2^8 samples in the MA filter the output is skewed, but we are still able to provide 42 ns RMS jitter and an extremely rapid setup time compared to our method described in Chapter 2. We divided the set-up time by 64 and increased the precision by a factor of 2. And the latency evolution does not show any discontinuity.

In Figure 4.19, we show that we are able to reach higher precision with a higher number of samples. With 2^{12} samples we are able to reduce the 1.8 µs jitter down to 90 ns of peak-to-peak jitter and 15 ns RMS jitter. With this filter, we are able high precision with a 4 ms setup time, a 16-fold improvement from the previous method.

Ideally, in order not to overload the network we might want to only turn on the DDX bus when traffic from time sensitive clients is present. So a rapid setup time is prevalent. This would steer us towards using the filters with a lower number of samples. In Figure 4.20 we show the peak-to-peak jitter of each filter for different network jitter. This gives us a chart for choosing the best filter according to the QoS the application needs and the amount of jitter expected on the network.

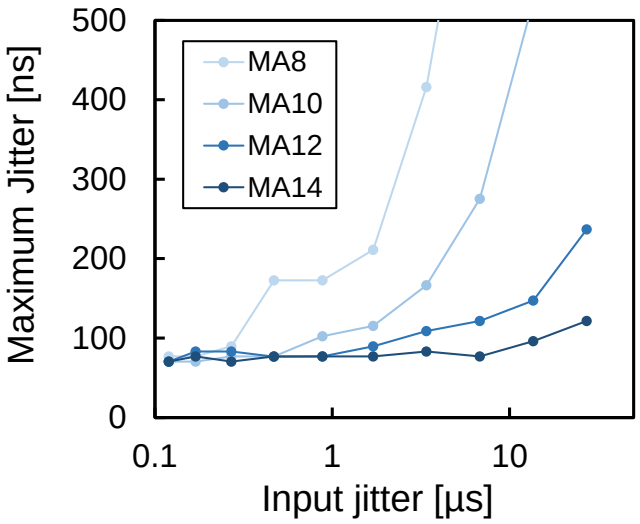

Figure 4.20: Maximum observed jitter after compensation for different network jitter.

We show that each filter possess a maximum of network jitter it can tolerate, and in the best case scenario, the MA filter with 2^{14} samples is able to reduce network jitter by a hundred folds.

Using the Moving Average filter in the remote clock tracking mechanism for jitter compensation is a great choice that enables a rapid setup time, under 10 ms, and offer a high precision when reducing network jitter by providing less than 100 ns peak-to-peak jitter and 15 ns RMS jitter.

4.4.2 Providing constant latency with the infinite impulse response filter

In this subsection, we show the performance of the jitter compensation mechanism with remote clock tracking using the Infinite Impulse Response (IIR) filter. In the

analytical study we saw that this filter should offer a better reduction of jitter at the cost of a longer setup time. And this choice offers a low memory implementation as no buffer is needed so if the memory is scarce in the device where this feature is implemented, our choice might focus on this method.

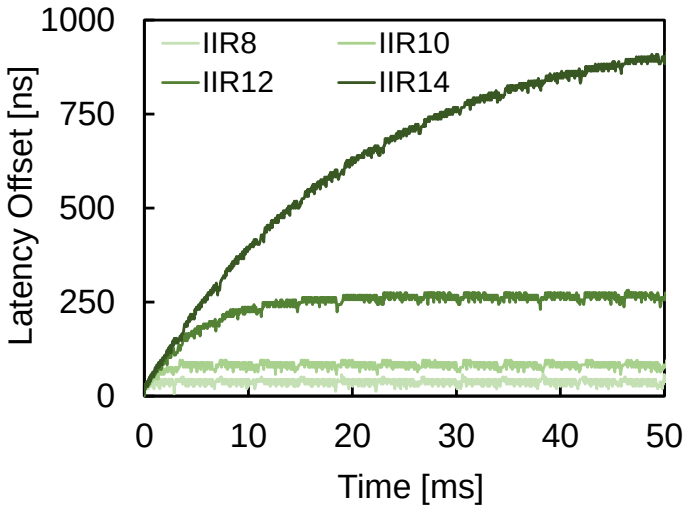

Figure 4.21: Infinite impulse response (IIR) filter response for following 50 ppm clock drift with no network jitter.

In Figure 4.21, we show the latency offset created during the setup time of the filter caused by the high frequency difference, 50 ppm, between the ingress and egress DDX add-on cards. We measure the latency of packets that we sent through the DDX bus right at the start of the setup. We choose the zero offset to be the latency of the first packet.

With no network jitter we can see the full and continuous evolution of the offset between the devices, the same evolution as described by equation 4.3. The setup times are longer than when using the MA filter by about a factor 4.

We continue to see periodic peaks in the evolution of the offset with any filter coefficient. This is due to the implementation. In these implementations, as I control the code of every device, I could deal with this issue but in a real scenario with application sending in their traffic, it would not be feasible to get rid of such asperities. So we chose to leave this issue as is.

In Figure 4.22 and Figure 4.23, we present the latency evolution and distribution of clients passing through the DDX bus once it has completed its setup. The bus is subjected to 1.8 µs peak-to-peak jitter.

In Figure 4.22, we show the performance of the 2^{-8} coefficient IIR filter for remote clock tracking. Compared to the equivalent MA filter which results are presented

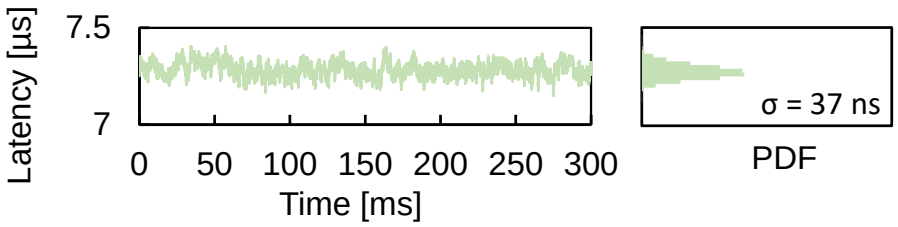

Figure 4.22: Latency evolution (over 300 ms) and distribution (over 1 minute)after compensation using Infinite Impulse Response (IIR) filter of coefficient 2^{-8} with 1.8 µs network jitter.

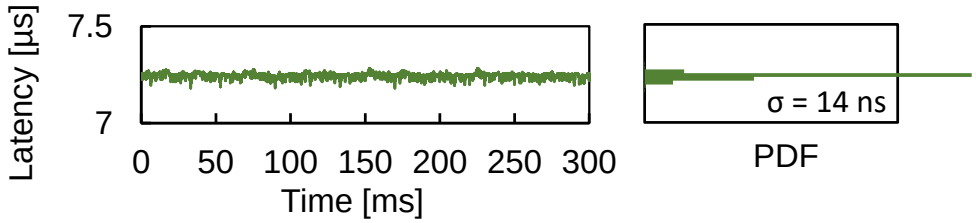

Figure 4.23: Latency evolution (over 300 ms) and distribution (over 1 minute) after compensation using Infinite Impulse Response (IIR) filter of coefficient 2^{-12} with 1.8 µs network jitter.

in Figure 4.18, this filter is able to achieve a better jitter reduction. This filter reduces a 1.8 µs peak-to-peak network jitter down to 160 ns of peak-to-peak jitter and achieving 37 ns RMS jitter.

In Figure 4.23, we show the performance of the 2^{-12} coefficient IIR filter for remote clock tracking. By increasing the setup time we are able to gain even more jitter reduction. This filter is able to reduce the 1.8 µs peak-to-peak jitter down to 83 ns peak-to-peak jitter and 14 ns RMS jitter. These performances are achieved at the price of a 20 ms setup time.

In Figure 4.24, we compare the performance of the IIR filter for various coefficient. We show the peak-to-peak jitter each filter is able to achieve given a certain network jitter.

The IIR filters show a better stability compared to their MA counterpart. They are able to maintain low jitter for a longer time. The IIR filter with 2^{14} coefficient is able to provide a maximum of 200-fold jitter reduction, though its setup time is longer, around 100 ms.

The Infinite Impulse Response is a great filter choice when performing remote clock tracking in a device which has low memory. Moreover, it is able to perform better jitter reduction than the Moving Average filter for equivalent coefficient. Its only downside is the setup time, 4 times greater than the MA filter.

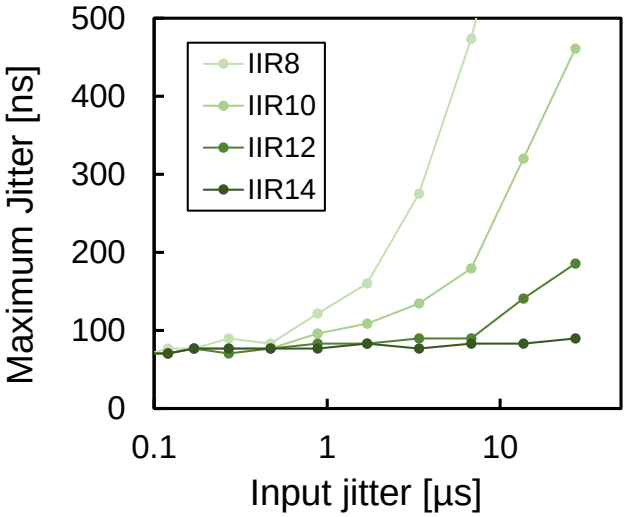

Figure 4.24: Maximum observed jitter after compensation for different network jitter.

4.4.3 Filter comparison

In this section we compare the results of both filter and provide our thoughts on how to choose a filter the perform our method of remote clock tracking.

In Figure 4.25, we compare the set-up time of the filter and different parameters and the maximum network they are able to cope with. We defined the maximum jitter as the maximum peak-to-peak network jitter they are able to take in whilst still providing peak-to-peak jitter under 100 ns.

Within each family of filter we can clearly see a trade-off between the setup time and the maximum jitter, which we have explore in the previous subsections. For a given filter coefficient the IIR is slightly better at reducing jitter than the MA filter but presents a 4 times longer setup time.

When choosing a filter for a time critical application and a estimated network jitter, based on the reserved path the DDX bus takes through the network. I would tend to favor the MA filter. The improvement in setup time overshadows the small improvement in jitter reduction. And a higher number of samples for the MA filter will often equalize the setup time of the IIR filter but improve on the jitter reduction. In Figure 4.25, we can see that the MA filter with 2^{10} samples has about the same setup time as the IIR filter of coefficient 2^{-8} and is able to compensate network jitter for a longer time. The same relation is found for other coefficient, when comparing MA filter of 2^n samples and IIR filter of coefficient $2^{-(n-2)}$.

Where the IIR filter gains terrain is in the implementation. Almost no memory besides the current value of the filter is required whereas the MA need to store M samples. For MA filter of 2^{12} samples, in our implementation we must include a

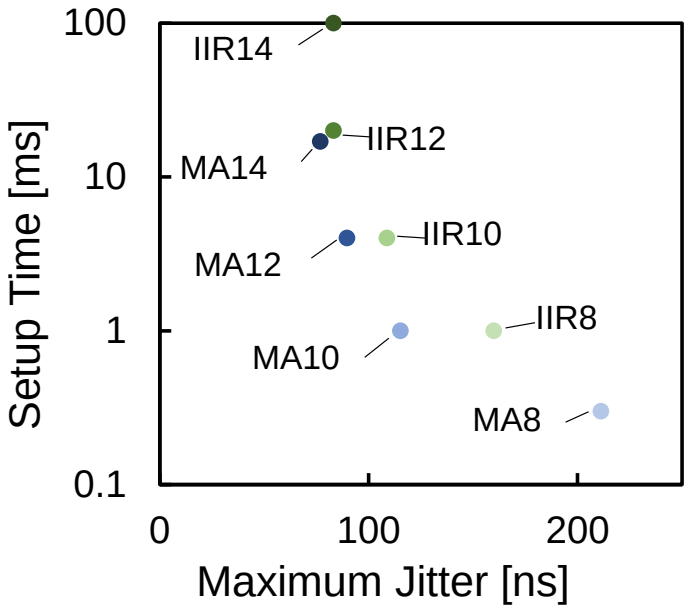

Figure 4.25: Comparison between setup time and maximum jitter for different parameter of moving average (MA) and infinite impulse response filter (IIR).

buffer of 81.9 kbit. Though we could reduce the width of the timestamp based on the network jitter, according to what we saw in Section 4.3.3.

4.5 Conclusion

In this chapter, we exposed a perverse effect of matching the ingress frequency through measurement at the egress node. We introduced a method for tracking the evolution of the offset between the ingress and egress node. This method not only improves the jitter for the time-critical clients, but also decreases the turn-up time when creating a new bus. It also allows for performance tuning when choosing a trade-off between ultra-low jitter and rapid turn-up time.

We computed and simulated different filters for tracking the offset. We studied their characteristics and response to our problem conditions. In a physical implementation of this new method we demonstrated setup time on the millisecond scale for a jitter reduction to a maximum of 100 ns.

Conclusion

In this manuscript, we have addressed the challenge of integrating deterministic and dynamic networks to the Information Technology network to support time-critical applications originating from telecommunications and Operation Technology networks.

To avoid the significant upfront costs of overhauling the entire network architecture to cater to a small fraction of the traffic, we proposed a pay-as-you-grow solution enhancing legacy network to provide deterministic and dynamic performance. We devised the DDX protocol as a solution to carry time-critical applications over Ethernet networks. By strategically placing network cards only in the locations where deterministic performance is required, we ensure a targeted and efficient upgrade.

Whilst multiple other project advocate for synchronized ring networks [7, 35] to provide deterministic performances, we have proposed a solution capable of fulfilling the requirements of applications originally deployed in dedicated OT networks without requiring a complete synchronization of the network.

Our approach is built around three essential aspects:

- The scheduling of time-critical clients,
- The selection and reservation of a path through the Ethernet network,
- A latency variation compensation process to reduce the remaining jitter.

This protocol allows us to provide strong deterministic performance without requiring network synchronization.

Since time-critical traffic constitutes only a small fraction of the overall traffic, even in the edge network, we focused on developing a solution with a limited impact on the network. We have highlighted the advantages of leveraging TSN features, like preemption, which is already being deployed in commercial products, to create the bridge necessary for our communication protocol.

A large part of the work presented in this manuscript, was dedicated to demonstrating our ideas experimentally over physical realistic networks using commercially available equipments. We have designed various essential building blocks to implement our protocol, on an FPGA based prototyping platform, to assess its performance. We have tested our prototype, in combination with standard Ethernet switches, and measured latency variation below 120 ns, with RMS jitter below 40 ns. In a second implementation, improving the continuity of the latency variation compensation process, we further improved the maximum latency variation and jitter. We have successfully demonstrated 80 ns maximum latency variation with 15 ns RMS jitter.

The second key aspect in catering to time-critical applications is *dynamicity*, allowing applications to be served a guaranteed latency shortly after requiring the resources. We have studied the set-up time of our equipment and demonstrated the creation of a deterministic bus in few milliseconds. We have highlighted the trade-off between the set-up time and the achievable jitter guarantee. We have showed that we are able to provide 40 ns RMS jitter with a set-up time of 0.3 ms, vastly fulfilling the requirements of time-critical applications like augmented distributed computing [14].

In the meantime, we have also developed a latency monitoring system. We have used the same strategic approach of placing monitoring points where they were needed and devised an accurate synchronization protocol enhancing the standard PTP protocol with local clock generator configurations to be tolerant to synchronization link failure. We have demonstrated a synchronization precision of 6.4 ns for a single link, equivalent to the period of the clock signal used. We have showed a linear evolution of the synchronization precision when going through successive links. This accurate monitoring system enabled us to gain insights on the behavior of our protocol and was used to improve the determinism of our transport protocol.

Perspectives

Our work, in this thesis, has been focused on creating and implementing a protocol and network of equipment to carry time critical applications in unsynchronized networks. We have showcased promising results that lead us to think that this technology could garner a wider audience. We can wonder what the next steps in promoting such technology can be.

We can head in three different directions to distribute the knowledge built. First we can ease the barrier of entry for developing and interacting with such technology. Second, we can demonstrate its integration to real life cases and its benefits. Lastly, we can continue to improve the technology to continue convincing people. These key points would be a stepping stone in being able to integrate our protocol in existing norms, ideally to be part of the TSN norms. We would need to reach out to the TSN taskforce to discuss which element can be carried into a norm and how we can improve to ease the integration.

In order to demonstrate our results we had to implement the DDX protocol on a hardware platform. Hardware description language and developing at the RTL level is hard and time-consuming. Still, it is necessary to have hardware, low level implementation for real-time traffic control and monitoring. In order for more people to collaborate on the project we could use High Level Synthesis (HLS) to build the project destined to an FPGA. HLS can be performed from C or C++ code which would enable a wider audience to collaborate including people developing network applications.

We thought about using tools and methodologies to automatically generate the hardware implementations as the P4 language, a programming language for controlling forwarding planes in network devices. This language can be useful for multiple tasks we have manually implemented, like adding information in the header, handling packets in designated FIFOs based on information present in the header. However, this language lacks the timing management. Contributing time-management features to the open-source P4 project would be a valuable addition, enabling the language to cater to applications with stringent timing requirements.

In this manuscript we have demonstrated the performance of our protocol over commercially available network equipment. We showed how we could reduce the jitter of traffic generated by a network tester. To generate greater interest and validate the potential of our technology, we should conduct a physical demonstration of the performance improvements it enables. While a study conducted by Sahoo [14] simulates the performance gains in shared computing brought by a deterministic network, a physical demonstration still remains to be performed.

In the early days of the thesis we had envisaged and started to contact a company selling a remote gaming service. In this setup, the game would run on a server located in an edge datacenter, streaming the game video to the user, who would then send back command inputs. For fast-reaction games, maintaining a high frame rate and ensuring minimal latency for player inputs are crucial for a smooth gaming experience. Ensuring an equal latency to all players could also add to the fairness of the game. This use-case presents an ideal opportunity to apply the DDX protocol, potentially leading to demonstrations with real-world network traffic and showcasing its effectiveness in handling time-critical applications. Preliminary studies showed promising results which interested both our partner and us, however the collaboration was not carried through.

We believe that we have reached the limits of remote time alignment within the DDX protocol, with only marginal precision improvements still possible. However, we think that we can improve the synchronization protocol we used for directly connected links. As we have access to the clock signal of the synchronization source we could continuously measure the time difference between the local oscillator and the remote clock, at a precision lower than the clock period, to estimate when the clocks become misaligned and correct for the time error. This would allow us to reduce the frequency of the PTP communication, freeing bandwidth for the reporting of statistics.

The work on HLS development simplification and the further improvement of the synchronization protocols could be carried out in new collaborations and thesis projects.

Glossary

Network : A communication network designates the equipment, the links and the control software that allow for the communication of data between two or several devices, machines or servers.

Latency : The latency is the (one-way) time of travel, measured in seconds, of data from source to destination through a network. When source and destination are the same, we call it Round Trip Time (RTT). If left unspecified latency refers to one-way latency.

Jitter : Jitter is the variation of latency. The norm RFC3393 [9] defines PDV as an alternative to jitter for Internet Protocol performance metric as the latency difference between two given packets. They avoid the term jitter because it has more than one signification. In this norm, the most common meaning of jitter is the variation of a signal with respect to a clock.

Often the concept of delay variation between two packets does not capture possible latency drifts. So, in this document we keep PDV for the latency difference between two consecutive packets, and we define jitter as the standard deviation of latency, measured in seconds.

In the clocking field, jitter can be measured in two ways, RMS jitter or peak-to-peak jitter. For our purpose, RMS jitter is the standard deviation of the distribution of latency. And peak-to-peak jitter is the difference between the maximum and minimum latency over a period of time. In this manuscript, peak-to-peak jitter is measured over periods of 100 ms.

Time-critical application : We call time-critical applications, applications that have stringent latency and 'jitter' requirements for the proper completion of the application. The application can require the data to arrive within a specified time window, or with the very similar latency as the previous packet. In this manuscript, we take interest in time-critical applications requiring latency inferior to 1ms and jitter inferior to 1µs.

Best-effort : Best-effort refers to traffic or application that have no need for controlled latency or guarantee of delivery. A mainstream example of a best-effort application is the streaming of on-demand video which can tolerate some loss (xx

Set-up time : We define the set-up time as the amount of configuration time it takes to go from an off or idle state to the first data delivery.

Traffic Flow : We call traffic flow all the information an application sends, often through Ethernet packet, from one point of the network to another. An application can generate multiple flows if there are multiple end destinations. In order to simplify we confuse traffic flow and application traffic, unless explicitly specified.

Flow Lifetime : The flow lifetime is the duration of the traffic flow. The flow lifetime can be infinite for example when an application sends periodic messages. Alternatively, it can be finite for a data transfer at the end of a calculation.

Determinist : Deterministic means exhibiting a strong performance guarantee according to attributes that need to be specified, often in terms of latency and jitter but not only. Deterministic can be applied to a transmission link, a service or to a piece of hardware or software. A deterministic network would guarantee delivery and provide the same latency for every packet of the same flow.

Dynamic : Dynamic is the ability of an infrastructure to react (to be reconfigured) according to the set-up and teardown of applications. Current Ethernet and IP network set a standard for dynamic timescales (xxµs), which deterministic networks should ideally replicate. In this work, we would like to engineer networks where set-up time is shorter than the application lifetime for most of the traffic flows (e.g. 90

Definition for the project

DDX node A DDX node is the aggregation point where time-critical services meets other services, possibly best-effort, in the network. A DDX node is the add-on card we will describe in this manuscript which is inserted in the network and has the ability to handle time-critical applications with the DDX protocol.

Ingress node : For a given time critical application, we call ingress node the DDX node where the information enters the bus.

Egress node : For a given time critical application, we call egress node the DDX node where the information leaves the bus.

Deterministic path : We call deterministic path the whole physical (i.e. optical fiber) and logical (i.e. switch) resources that are reserved between an ingress node and an egress node to fulfill specified performance guarantees. Here the performance

guarantees amount primarily to providing an upper-bounded latency, as required by time critical applications.

Intermediate node : The intermediate node is a DDX node in the middle of the bus where traffic can both enter and exit the bus.

DDX slot : A DDX slot is the smallest time slot that can be reserved. All slots have the same duration and form a time division multiplexed bus. In our scenario, one DDX slot can only host one time critical application and applications can reserve multiple slots.

DDX packet : The DDX packet is an Ethernet packet containing timing information and slot information and time sensitive payload, which is sent during one DDX slot. The size of a DDX packet can vary depending on the payload, and the residual space during the DDX slot can be used to accommodate best-effort applications.

Hardware definition

FPGA : An Field Programmable Gate Array is a configurable and reprogrammable integrated circuit used to perform parallel and rapid tasks. They offer a compromise between a dedicated integrated circuit which very efficiently and rapidly performs the task it was developed for, and the flexibility of a microprocessor executing a linear set of instructions.

DCO : An oscillator frequency is often controlled through voltage supply. A Digitally Controlled Oscillator offers the possibility to digitally control the voltage allowing to fine tune the oscillator frequency.

References

[1] Cisco, Ed., *Cisco annual internet report (2018–2023) white paper*, Mar. 9, 2020.

[2] A. Tzanakaki, M. P. Anastasopoulos, and D. Simeonidou, "Converged optical, wireless, and data center network infrastructures for 5g services," *Journal of Optical Communications and Networking*, vol. 11, no. 2, A111, Feb. 1, 2019, ISSN: 1943-0620, 1943-0639. DOI: 10.1364/JOCN.11.00A111.

[3] M. Hermann, T. Pentek, and B. Otto, "Design principles for industrie 4.0 scenarios," in *2016 49th Hawaii International Conference on System Sciences (HICSS)*, Koloa, HI, USA: IEEE, Jan. 2016, pp. 3928–3937, ISBN: 978-0-7695-5670-3. DOI: 10.1109/HICSS.2016.488.

[4] X. Jin *et al.*, "Optimizing bulk transfers with software-defined optical WAN," in *Proceedings of the 2016 ACM SIGCOMM Conference*, Florianopolis Brazil: ACM, Aug. 22, 2016, pp. 87–100, ISBN: 978-1-4503-4193-6. DOI: 10.1145/2934872.2934904.

[5] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, Hong Kong China: ACM, Aug. 27, 2013, pp. 15–26, ISBN: 978-1-4503-2056-6. DOI: 10.1145/2486001.2486012.

[6] M. A. Mestre, G. De Valicourt, P. Jenneve, H. Mardoyan, S. Bigo, and Y. Pointurier, "Optical slot switching-based datacenters with elastic burst-mode coherent transponders," in *2014 The European Conference on Optical Communication (ECOC)*, Cannes, France: IEEE, Sep. 2014, pp. 1–3, ISBN: 978-2-9549444-0-1. DOI: 10.1109/ECOC.2014.6963853.

[7] B. Pan, X. Xue, X. Guo, and N. Calabretta, "Precise time distribution and synchronization for low latency slotted optical metro-access networks," *Journal of Lightwave Technology*, vol. 40, no. 8, pp. 2244–2253, Apr. 15, 2022, ISSN: 0733-8724, 1558-2213. DOI: 10.1109/JLT.2021.3138456.

[8] W. Lautenschlaeger, L. Dembeck, and U. Gebhard, "Prototyping optical ethernet—a network for distributed data centers in the edge cloud," *Journal of Optical Communications and Networking*, vol. 10, no. 12, p. 1005, Dec. 1, 2018, ISSN: 1943-0620, 1943-0639. DOI: 10.1364/JOCN.10.001005.

[9] C. Demichelis and P. Chimento, "IP packet delay variation metric for IP performance metrics (IPPM)," RFC Editor, RFC3393, Nov. 2002, RFC3393. DOI: 10.17487/rfc3393.

[10] A. Lechler and A. Verl, "Software defined manufacturing extends cloud-based control," in *Volume 3: Manufacturing Equipment and Systems*, Los Angeles, California, USA: American Society of Mechanical Engineers, Jun. 4, 2017, V003T04A025, ISBN: 978-0-7918-5074-9. DOI: 10.1115/MSEC2017-2656.

[11] 3GPP, *Study on communication for automation in vertical domains (CAV)*, Jul. 11, 2020.

[12] E. Grossman, *Deterministic networking use cases RFC 8578*, May 10, 2019.

[13] *V2x white paper v 1.0*, in collab. with N. V. T. Force, Jun. 17, 2018.

[14] S. Sahoo, S. Bigo, and N. Benzaoui, "Introducing best-in-class service level agreement for time-sensitive edge computing," in *2021 European Conference on Optical Communication (ECOC)*, Bordeaux, France: IEEE, Sep. 13, 2021, pp. 1–4, ISBN: 978-1-66543-868-1. DOI: 10.1109/ECOC52684.2021.9605824.

[15] Y. Ebihara *et al.*, "Tele-assessment of bandwidth limitation for remote robotics surgery," *Surgery Today*, vol. 52, no. 11, pp. 1653–1659, Nov. 2022, ISSN: 0941-1291, 1436-2813. DOI: 10.1007/s00595-022-02497-5.

[16] 5. Americas, *New services & applications with 5g ulrta-reliable low latency communications*, Nov. 2018.

[17] *Common public radio interface (CPRI); interface specification*, in collab. with Ericsson AB, Huawei Technologies Co. Ltd, NEC Corporation, Alcatel Lucent, and Nokia Siemens Networks GmbH & Co. KG., Aug. 30, 2013.

[18] *Common public radio interface: eCPRI presentation*, in collab. with Ericsson AB, Huawei Technologies Co. Ltd, Nokia, and NEC Corporation, Jun. 22, 2018.

[19] *Common public radio interface: eCPRI interface specification*, in collab. with Ericsson AB, Huawei Technologies Co. Ltd, Nokia, and NEC Corporation, Jun. 25, 2018.

[20] H.-K. Tan, R. Radhakrishna Pillai, C.-K. Tham, L. Wong, and J. Biswas, "Impact of ATM cell delay on multimedia applications," *Computer Communications*, vol. 23, no. 13, pp. 1215–1222, Jul. 2000, ISSN: 01403664. DOI: 10.1016/S0140-3664(00)00199-7.

[21] "IEEE standard for local and metropolitan area networks–timing and synchronization for time-sensitive applications," IEEE, Sep. 28, 2017, ISBN: 9781504464307. DOI: 10.1109/IEEESTD.2020.9121845.

[22] "IEEE std 1588-2008, IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," p. 289,

[23] *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): IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendme.* IEEE., 2016, OCLC: 1127165364, ISBN: 978-1-5044-0721-2.

[24] L. Velasco and M. Ruiz, "Supporting time-sensitive and best-effort traffic on a common metro infrastructure," *IEEE Communications Letters*, vol. 24, no. 8, pp. 1664–1668, Aug. 2020, ISSN: 1089-7798, 1558-2558, 2373-7891. DOI: 10.1109/LCOMM.2020.2989080.

[25] L. Levrau and D. Remedios, "Guidelines for a cost optimised 5g WDM-based fronthaul network," in *2021 European Conference on Optical Communication (ECOC)*, Bordeaux, France: IEEE, Sep. 13, 2021, pp. 1–4, ISBN: 978-1-66543-868-1. DOI: 10.1109/ECOC52684.2021.9605805.

[26] W. Lautenschlaeger and A. Francini, "Global synchronization protection for bandwidth sharing TCP flows in high-speed links," in *2015 IEEE 16th International Conference on High Performance Switching and Routing (HPSR)*, Budapest, Hungary: IEEE, Jul. 2015, pp. 1–8, ISBN: 978-1-4799-9871-5. DOI: 10.1109/HPSR.2015.7483103.

[27] S. Bigo, N. Benzaoui, K. Christodoulopoulos, R. Miller, W. Lautenschlaeger, and F. Frick, "Dynamic deterministic digital infrastructure for time-sensitive applications in factory floors," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 27, no. 6, pp. 1–14, Nov. 2021, ISSN: 1077-260X, 1558-4542. DOI: 10.1109/JSTQE.2021.3093281.

[28] *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014): IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption*. Place of publication not identified: IEEE, 2016, OCLC: 972631006, ISBN: 978-1-5044-2257-4.

[29] K. Jeffay, D. Stanat, and C. Martel, "On non-preemptive scheduling of period and sporadic tasks," in *[1991] Proceedings Twelfth Real-Time Systems Symposium*, San Antonio, TX, USA: IEEE Comput. Soc. Press, 1991, pp. 129–139, ISBN: 978-0-8186-2450-6. DOI: 10.1109/REAL.1991.160366.

[30] K. Christodoulopoulos *et al.*, "Enabling the scalability of industrial networks by independent scheduling domains," in *Optical Fiber Communication Conference (OFC) 2020*, San Diego, California: Optica Publishing Group, 2020, Th2A.24, ISBN: 978-1-943580-71-2. DOI: 10.1364/OFC.2020.Th2A.24.

[31] *IEEE Std 802.3-2018, IEEE Standard for Ethernet*. Place of publication not identified: IEEE, 2018, OCLC: 1056778434, ISBN: 978-1-5044-5090-4 978-1-5044-5798-9.

[32] "Si5341/40 rev d data sheet," in collab. with Skyworks,

[33] T. Corporation, *7m48072002 48mhz quartz datasheet*.

[34] "IEEE standard for local and metropolitan area networks – time-sensitive networking for fronthaul," IEEE, ISBN: 9781504449090. DOI: 10.1109/IEEESTD.2018.8376066.

[35] N. Benzaoui *et al.*, "CBOSS: bringing traffic engineering inside data center networks," *Journal of Optical Communications and Networking*, vol. 10, no. 7, B117, Jul. 1, 2018, ISSN: 1943-0620, 1943-0639. DOI: 10.1364/JOCN.10.00B117.

[36] J. Joung and J. Kwon, "Zero jitter for deterministic networks without time-synchronization," *IEEE Access*, vol. 9, pp. 49 398–49 414, 2021, ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3068515.

[37] *Operation, administration and maintenance (OAM) functions and mechanisms for ethernet based networks*, 2015.

[38] A. Gulenko, M. Wallschlager, and O. Kao, "A practical implementation of in-band network telemetry in open vSwitch," in *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, Tokyo: IEEE, Oct. 2018, pp. 1–4, ISBN: 978-1-5386-6831-3. DOI: 10.1109/CloudNet.2018.8549431.

[39] R. Joshi, T. Qu, M. C. Chan, B. Leong, and B. T. Loo, "BurstRadar: practical real-time microburst monitoring for datacenter networks," in *Proceedings of the 9th Asia-Pacific Workshop on Systems*, Jeju Island Republic of Korea: ACM, Aug. 27, 2018, pp. 1–8, ISBN: 978-1-4503-6006-7. DOI: 10.1145/3265723.3265731.

[40] A. Nasrallah *et al.*, "Ultra-low latency (ULL) networks: the IEEE TSN and IETF DetNet standards and related 5g ULL research," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 1, pp. 88–145, 2019, ISSN: 1553-877X, 2373-745X. DOI: 10.1109/COMST.2018.2869350.

[41] L. Liao, V. C. M. Leung, and M. Chen, "An efficient and accurate link latency monitoring method for low-latency software-defined networks," *IEEE Transactions on Instrumentation and Measurement*, vol. 68, no. 2, pp. 377–391, Feb. 2019, ISSN: 0018-9456, 1557-9662. DOI: 10.1109/TIM.2018.2849433.

[42] A. Atary and A. Bremler-Barr, "Efficient round-trip time monitoring in OpenFlow networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, San Francisco, CA, USA: IEEE, Apr. 2016, pp. 1–9, ISBN: 978-1-4673-9953-1. DOI: 10.1109/INFOCOM.2016.7524501.

[43] D. Mills, "Network time protocol (NTP)," RFC Editor, RFC0958, Sep. 1985, RFC0958. DOI: 10.17487/rfc0958.

[44] D. Mills, J. Martin, J. Burbank, and W. Kasch, "Network time protocol version 4: protocol and algorithms specification," RFC Editor, RFC5905, Jun. 2010, RFC5905. DOI: 10.17487/rfc5905.

[45] ITU, *G.8262 : timing characteristics of a synchronous equipment slave clock*, Nov. 29, 2018.

[46] C. Kim *et al.*, "In-band network telemetry (INT)," p. 28,

[47] M. Lipinski, T. Wlostowski, J. Serrano, and P. Alvarez, "White rabbit: a PTP application for robust sub-nanosecond synchronization," in *2011 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*, Munich, Germany: IEEE, Sep. 2011, pp. 25–30, ISBN: 978-1-61284-893-8. DOI: 10.1109/ISPCS.2011.6070148.

[48] Xilinx, *10g ethernet PCS/PMA v6.0 product guide (PG068)*, Feb. 4, 2021.

[49] Xilinx, *10g ethernet MAC v15.1 product guide (PG072)*, Jun. 6, 2018.

Appendix

A Experimental Setup

In this appendix we will describe all the experimental set-up that were used during this thesis.

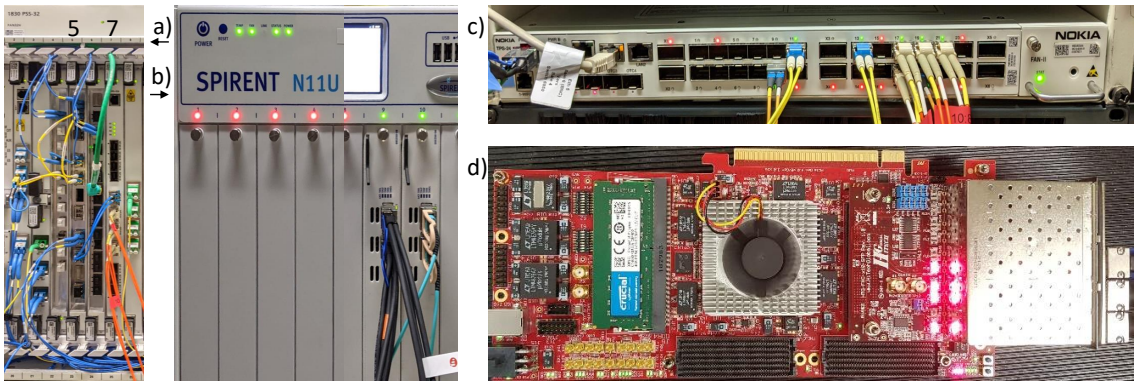

Figure A.1: a) OTN switch set-up : Nokia 1830 Photonic Service Switch (PSS), b) Traffic generator analyzer : Spirent N11U, c) Ethernet switch : Nokia 1830 TPS, d) FPGA development board : HTG930

A.1 Common equipment to all experiments

FPGA

Most often when referring to the FPGA in the experiment we actually mean a development board. On a development board, the main component is the FPGA, but many other chips and connector are on the board allowing a wide range of applications.

The development board we chose was the HTG930, a board designed for high-speed communications, with a *Xilinx XCVU190 FPGA* at its core. This FPGA has 52 gigabit transceiver routed on the board to three *FMC+* port connectors. Of these ports we were able to connect only two 10 SFP+ mezzanine cards due to two *FMC+*

connector being to close to one another to fit the mezzanine board next to each other without using and extender.

In excerpt *d* of figure A.1, we can see a picture of the *HTG930* with one mezzanine board and two *FMC+* connectors left open.

Switch

For routing the traffic through an Ethernet network, we had two 2×12 *1830 TPS* switch, a recent Nokia switch that incorporate Time Sensitive Networking protocols. The two units of one switch can be used as two independent switches. In excerpt *c* of figure A.1, we can see a picture of the *TPS* switch with *SFP+* optical connector plugged on the two sub-switches.

Optical Transport Network

For routing the traffic using OTN technologies, we had two Photonic Service Switch (PSS) interconnected with a 5 km optical fiber spool. The PSS is divided in two parts, a layer Ethernet to frame the data into the OTN format, and a long haul optical card. The PSS can be seen in excerpt *a* of figure A.1 with the Ethernet switch in slot 7 and the long-haul optical card in slot 5.

Traffic Generator/Analyzer

To generate and monitor traffic we had a *Spirent N11U* crate with a card that has four *QSFP* connector each running at 100 Gbps maximum. In all our configuration we only used a 40 Gbps data rate and *QSFP* adaptor to four *SFP+* connector to obtain four 10 Gbps traffic flows.

We were able to configure the *Spirent* through its software allowing us to program each traffic flow with unique source and destination, protocols, frame length, patterns and data rate.

In excerpt *b* of figure A.1, we can see a picture of the *Spirent N11U* with a 4 *QSFP+* board inserted. We also see the beginning of the *QSFP+* to 4 *SFP+* connector that we had to connect to the switches and FPGA.

A.2 Ethernet switch experimental set-up

In the first experiment, we measure the latency of time critical traffic through Ethernet switches either configured using VLAN or preemption. In Figure A.2 we can see the experimental setup. This set-up is composed of one traffic generator/analyzer, two FPGA development boards, and switches with two distinct units.

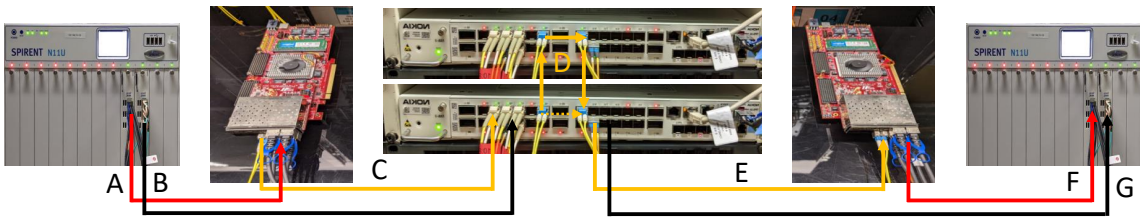

Figure A.2: Ethernet switches experimental setup

All optical connectors allow a maximum 10Gbps throughput, the data rates mentioned in the following description is an average data rate seen on the described flows. The traffic flows can be described as follows:

- **A:** Four 1Gbps constant bit rate, 256 bytes Ethernet frames (Time critical clients).
  Source: Traffic generator, Destination: FPGA development board.
- **B:** Four 1Gbps flows, burst of 10 1518 bytes Ethernet frames (Best-effort clients).
  Source: Traffic generator, Destination: switch.
- **C:** One DDX bus, here approximately 4Gbps. Each client has one slot reserved in a window of 8 slots.
  Source: FPGA development board, Destination: switch.
- **D:** One 10Gbps capable network port routed through two (dotted arrow) or four switches (plain arrows).*
  Source: Switch, Destination: Switch.
- **E:** DDX bus egress.
  Source: Switch, Destination: FPGA development board.
- **F:** Egress of the four time-critical clients.
  Source: FPGA development board, Destination: Traffic analyzer.
- **G:** Egress of the four Best-effort clients (only in preemption).
  Source: Switch, Destination: Traffic analyzer.

*There are two configurations of the switch in the middle of the experimental-setup.

The first experimental scenario is 2S-VLAN Only frames with a specific VLAN tag are allowed to enter the network port described by flow **D**. In this configuration, only the traffic flow **C** is tagged with the VLAN making it the only traffic allowed

in flow **D**. This is the first experimental scenario. We ran this configuration with both two and four switches.

The second experimental scenario is 4S-PREEMPT. In this scenario, the four switches are configured with preemption. We specify in the configuration of the switch which port is express and which is normal. We configure the destination port of flow **C** to be express and the destination of flow **B** to be normal.

A.3 Optical Transport Network (OTN) experimental set-up

In a second experiment, OTN-shared, we measure the latency of time critical clients when scheduled by DDX in an Optical Transport Network (OTN). In Figure A.3, we can see the experimental setup. This setup is comprised of a traffic generator/analyzer, two FPGA development boards, and two OTN nodes separated by a 5 km optical fiber spool.

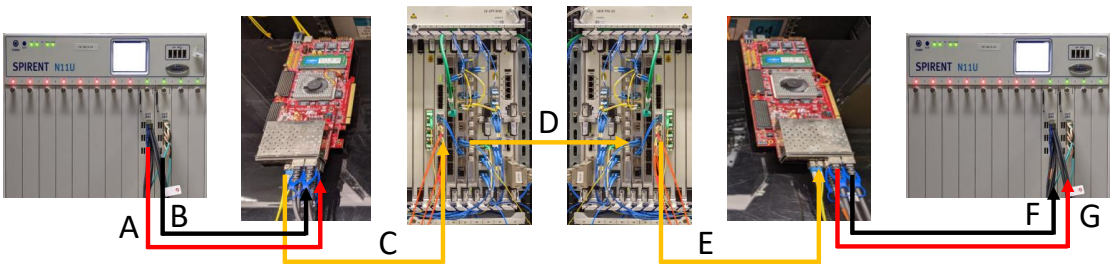

Figure A.3: DDX bus through Optical transport network (OTN)

The traffic flows can be described as follows:

- **A:** Two 1Gbps constant bit rate, 256 bytes Ethernet frames (Time critical clients).
  Source: Traffic generator, Destination: FPGA development board.
- **B:** One 1Gbps flows, burst of 10 1518 bytes Ethernet frames (Best-effort client).
  Source: Traffic generator, Destination: FPGA development board.
- **C:** One DDX bus, here approximately 3 Gbps. Each client has one slot reserved in a window of 8 slots. The best-effort client is also given a slot
  Source: FPGA development board, Destination: OTN node.
- **D:** DDX bus inserted inside a 10 Gbps OTN container and sent to the second OTN node. The link between OTN nodes is a 100 Gbps link over 5 km optical fiber spool.
  Source: OTN node, Destination: OTN node.

- **E:** DDX bus egress.
  Source: OTN node, Destination: FPGA development board.
- **F:** Egress of the best-effort client.
  Source: FPGA development board, Destination: Traffic analyzer.
- **G:** Egress of the two time-critical clients.
  Source: FPGA development board, Destination: Traffic analyzer.

ECOLE
DOCTORALE

Titre : Garantie de service de bout en bout pour les réseaux optiques haut-débit

Mots clés : Garantie de service, Service bout-en-bout, Réseaux optiques, Variation de latence

Résumé : Poussée par un besoin croissant de bande passante et de performance, le réseau informatique s'est développé de telle sorte que les réseaux OT et de télécommunications cherchent à exploiter cette infrastructure pour leur expansion. Ces trois secteurs ont historiquement été séparés en raison de différentes exigences entre autre en matière de latence, de sa variation et de fiabilité. Pour répondre aux besoins des applications temps critiques, le groupe de travail Time Sensitive Network a développé de nouveaux ensembles de protocoles qui commencent à être mis en œuvre dans des produits commerciaux. D'autres groupes ont proposé des architectures novatrices avec contrôle du temps pour permettre des performances garanties, entre et à l'intérieur des centres de données périphériques. Dans ma thèse, je propose une solution pour transporter les applications temps critiques dans les réseaux existants, car elle ne nécessite pas de changer toute l'architecture. Je montre les avantages de sa mise en œuvre dans les réseaux TSN pour une solution pérenne avec une utilisation améliorée des ressources. Pour transporter le trafic temps critique dans les réseaux existants, je propose de créer un chemin en isolant et en planifiant le trafic temps critique sur un canal avec une latence garantie. Avec cette construction, je développe un algorithme pour effectuer une compensation de la variation de latence, permettant une transmission à latence constante pour le trafic temps critique. Dans un second temps, je propose une méthode de synchronisation et mets en œuvre un réseau de mesure principalement utilisé ici pour la mesure de la latence, m'aidant à obtenir des informations sur la distribution de la latence que mon protocole crée. Enfin, avec un algorithme de compensation de variation de latence amélioré, je démontre de meilleures performances en matière de gigue et étudie le temps de mise en service de notre protocole, permettant l'utilisation des ressources uniquement lorsque le trafic temps critique est présent. Dans ma thèse, je démontre, avec une implémentation FPGA, la réduction de la variation de latence, permettant aux applications des réseaux OT et de télécommunications de fonctionner sur les réseaux existants et augmentés par des normes TSN.

Title : End-to-end service guarantee for high-speed Optical networks

Keywords : Service guarantee, End-to-end services, Optical networks, Latency variation

Abstract : Driven by an ever-growing bandwidth and performance need, the IT network has grown such that OT and telecommunications networks are looking to exploit this infrastructure for their expansion. These three sectors have historically been separated due to different requirements, on latency, its variation and on reliability. To answer to time critical application needs, the Time Sensitive Network taskforce has developed new sets of protocols that are starting to be implemented in commercial products. Other groups have proposed novel architecture with time control to enable guaranteed performances between and inside edge datacenters. In my PhD I propose a solution to carry time critical application in legacy networks as it does not require to change the whole architecture. I show the benefits of its implementation in TSN networks for a future-proof solution with improved resource usage. To carry time critical traffic in legacy I propose to create a path by isolating and scheduling time critical traffic on a channel with guaranteed latency. With this construction, I build an algorithm to perform latency variation compensation enabling a constant latency transmission for time critical traffic. In a second time, propose a synchronization scheme and implemented a monitoring network primarily used here for latency monitoring, helping me gain insights on the distribution of latency that my protocol creates. Lastly, with an improved latency compensation algorithm, I demonstrate better jitter performances and study the turn-up time for our protocol enabling resource usage only when time-critical traffic is present. In my PhD I demonstrate, with an FPGA implementation and commercial product, latency variation reduction enabling OT and telecommunications network applications to run on legacy and TSN augmented network.

Institut Polytechnique de Paris
91120 Palaiseau, France