Universidad Politécnica
de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Grado en Grado en ingeniería informática

Trabajo Fin de Grado

Diseño de un sistema de gestión y entrega de tiempos para la red cuántica MadQCI-UPM

Autor: Daniel Fernández Calvo
Tutor: Alberto Juan Sebastian Lombraña

Madrid, JUNIO - 2025

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado
Grado en Grado en ingeniería informática

Título: Diseño de un sistema de gestión y entrega de tiempos para la red cuántica MadQCI-UPM

JUNIO - 2025

Autor: Daniel Fernández Calvo
Tutor: Alberto Juan Sebastian Lombraña
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Escuela Técnica Superior de Ingenieros Informáticos
Universidad Politécnica de Madrid

Índice general

Índice de Figuras

Índice de Tablas

Índice de Listings

Resumen

El presente Trabajo de Fin de Grado aborda el estudio, desarrollo y diseño de una plataforma de sincronización temporal para la red cuántica metropolitana MadQCI.

Se parte de una explicación detallada de los mecanismos de sincronización temporal en redes distribuidas, incluyendo tecnologías como NTP, PTP y White Rabbit. Después se explica la red MadQCI donde se va a desplegar este sistema, asumiendo las suposiciones que cumple la red y explicando los requisitos de la misma, tanto funcionales como no funcionales.

A continuación, se explican las diferentes arquitecturas utilizadas para cada uno de los protocolos específicos y se propone una arquitectura para los futuros nodos de la red.

Posteriormente, se proponen diferentes sistemas de entrega temporal para los usuarios de la red, como puedan ser una API REST o *anything as a service* (XaaS), mostrando una pequeña demo de cómo funcionarían.

Por último, se experimenta en un entorno virtual las posibilidades de servicio en un sistema de pruebas virtualizado y se hace una comparativa de los resultados obtenidos.

Palabras clave: NTP, PTP, QKD, White Rabbit, tiempo, sincronía

Abstract

This Final Degree Project addresses the study, development, and design of a time synchronization platform for the MadQCI metropolitan quantum network.

It begins with a detailed explanation of time synchronization mechanisms in distributed networks, including technologies such as NTP, PTP, and White Rabbit. The MadQCI network, where this system will be deployed, is then introduced, along with the assumptions it meets and both its functional and non-functional requirements.

Next, the different architectures used for each specific protocol are described, and a proposed architecture for future nodes of the network is presented.

Subsequently, several time delivery systems for network users are proposed, such as a REST API or *anything as a service* (XaaS), including a small demo to illustrate how they would work.

Finally, a virtual test environment is used to experiment with the service delivery system, and a comparison of the results obtained is carried out.

Keywords:NTP, PTP, QKD, White Rabbit, time, synchronization

Capítulo 1

Introducción

Este capítulo introduce la motivación, el contexto, los objetivos y la estructura del trabajo, proporcionando una visión general del proyecto y sus fundamentos.

1.1. Motivación del proyecto

Todos los sistemas en red deben estar sincronizados para poder prestar sus servicios. Esta sincronización puede ser necesaria para ejecutar sus procesos de forma síncrona, compartiendo una señal de reloj, o para tener un tiempo y fecha comunes que entregar a sus programas informáticos. Las redes cuánticas y, en especial, las de criptografía cuántica, no son una excepción.

Los sistemas de distribución cuántica de clave (QKD por sus siglas en inglés) permiten generar claves seguras entre nodos, para ser usadas en distintas aplicaciones criptográficas. La sincronía puede ser fundamental tanto para sostener las funcionalidades QKD como la de las aplicaciones, especialmente cuando se integran dispositivos de distintos fabricantes. Esto no solo garantiza que las claves se generen y distribuyan de forma coherente en los sistemas QKD, sino que también permite a los sistemas de control y gestión de la red operar la red de forma efectiva.

Las redes cuánticas no sólo integran sistemas cuánticos como los QKD, sino otros subsistemas que permiten su operación. Son de especial interés los sistemas de gestión de clave (KMS), que deben coordinar adecuadamente los eventos entre nodos. La falta de coherencia temporal puede dar lugar a errores en la validación, el uso o el almacenamiento de claves criptográficas, lo que compromete tanto la seguridad como la eficiencia de la red. Por ello, disponer de una capa de sincronización homogénea resulta fundamental para asegurar la operatividad del sistema en su conjunto.

Además, la red no solo cuenta con dispositivos de distribución cuántica de claves y los sistemas de control y gestión, sino que también mantiene una infraestructura de servicios clásicos destinada a facilitar la gestión, la recogida de métricas y datos, el control de acceso, así como la realización de experimentos. Esta red de apoyo está diseñada para que investigadores de distintas disciplinas puedan utilizar la plataforma y explorar aplicaciones relacionadas con las tecnologías cuánticas. En la mayoría de estos experimentos, la captura precisa de métricas temporales y la correcta secuenciación de los eventos resulta crucial para su éxito. Asimismo, los sistemas de control

de accesos y seguridad pueden beneficiarse de esta coherencia temporal. Al estar distribuida en nodos distintos, la sincronización del sistema debe estar centralizada y orientada a simplificar el trabajo de los investigadores.

1.2. Contexto del proyecto

Este TFG se desarrolla en el ámbito de las comunicaciones cuánticas y, en especial, de la criptografía cuántica. Existen tecnologías cuánticas, como la informática cuántica, que amenazan algunas de las primitivas criptográficas fundamentales hoy en día, como son los acuerdos de clave. La criptografía cuántica, por el contrario, propone una primitiva, la QKD, con seguridad incondicional.

En la actualidad, la ciberseguridad es uno de los principales retos para la ciudadanía, la industria y las instituciones. Cada día se ven comprometidos datos sensibles debido a ciberataques y a la actividad de ciberdelincuentes. Tradicionalmente, la seguridad de la información se ha garantizado mediante aplicaciones criptográficas que aseguran su confidencialidad, integridad y autenticación en el uso y acceso. Para sustentarlas, es necesario establecer secretos compartidos entre las partes, lo que históricamente se ha resuelto mediante el intercambio de claves de Diffie–Hellman [1] o mediante esquemas de cifrado de clave pública como RSA [2].

Estas primitivas criptográficas se fundamentan en problemas matemáticos que son computacionalmente inviables para los ordenadores clásicos. No poseen una seguridad incondicional, debido a que no es posible demostrar su seguridad ante un atacante que posea recursos infinitos. Por ello, el desarrollo del algoritmo de Shor [3] ha supuesto una amenaza significativa para estos sistemas criptográficos. Este algoritmo cuántico permite factorizar números enteros de forma eficiente en un ordenador cuántico funcional, lo que facilitaría romper la seguridad de muchos esquemas actuales como el cifrado RSA; otros algoritmos cuánticos conocidos pueden romper el intercambio de clave Diffie-Hellman.

Ante este posible escenario, la industria se ha visto obligada a investigar y desarrollar nuevos enfoques criptográficos resistentes a la computación cuántica, como los algoritmos post-cuánticos y la distribución cuántica de claves. Los primeros proponen intercambios de clave que se demuestran inmunes a los algoritmos cuánticos conocidos —p. ej., se denominan "*shor-resistant*"—, pero no se puede garantizar su seguridad incondicional, por lo que puede que se descubran algoritmos cuánticos que los rompan. Son, sin embargo, fáciles de desplegar en el corto plazo y baratos, por lo que son la opción propuesta por los reguladores. La QKD, por el contrario, se fundamenta en la Mecánica Cuántica y, por ello, sí posee una seguridad incondicional, por lo que sería una solución más costosa pero adecuada en el medio y largo plazo.

La red MadQCI [4] es un entorno experimental de pruebas en comunicaciones cuánticas instalado en Madrid por actores como la UPM, REDIMadrid o Telefónica Innovación Digital. Su funcionalidad principal es integrar en una misma red dispositivos de distribución cuántica de claves de diferentes fabricantes con tecnologías clásicas habituales en redes en producción. Esta infraestructura debe ser capaz de coordinar el funcionamiento de todos estos sistemas, que utilizan protocolos y mecanismos de entrega de datos distintos, lo que plantea un importante desafío de interoperabilidad y sincronización.

1.3. Objetivos

- Estudiar los fundamentos teóricos necesarios para comprender la sincronización de tiempos en redes, incluyendo protocolos clásicos como NTP y PTP, así como tecnologías avanzadas como *White Rabbit*, con especial atención a su aplicabilidad en redes cuánticas.
- Analizar el funcionamiento de la red MadQCI, identificando los dispositivos implicados, sus necesidades temporales y los requisitos técnicos para una sincronización precisa y coordinada entre nodos heterogéneos.
- Diseñar una arquitectura distribuida para la gestión y entrega de tiempos, que permita proporcionar una referencia temporal común a todos los nodos de la red, independientemente del fabricante o tecnología utilizada.
- Integrar distintas tecnologías de sincronización en la arquitectura propuesta, evaluando criterios como precisión, compatibilidad, coste y escalabilidad, y proponiendo una combinación óptima para el entorno MadQCI.
- Definir los mecanismos de entrega del servicio de tiempo a los usuarios o sistemas de la red, mediante interfaces como API REST, arquitecturas tipo *anything as a service* (XaaS) o protocolos estándar, garantizando accesibilidad y consistencia temporal.
- Evaluar el sistema propuesto a través de métricas relevantes, simulaciones o pruebas reales, y presentar los resultados obtenidos junto con las limitaciones detectadas y propuestas de mejora futura.

1.4. Estructura del Documento

Este documento se estructura en ocho capítulos, más una sección de bibliografía y varios anexos. A continuación se describe brevemente el contenido de cada uno de ellos:

- **Capítulo 1: Introducción**. Presenta el contexto general del trabajo, su motivación, los objetivos planteados y una visión global del contenido del documento.
- **Capítulo 2: Fundamentos teóricos**. Revisa los conceptos clave relacionados con la sincronización de tiempos en redes, incluyendo los protocolos clásicos (NTP, PTP), tecnologías avanzadas como White Rabbit, y su aplicación en redes cuánticas como QKD.
- **Capítulo 3: Análisis del sistema MadQCI-UPM**. Describe la red experimental MadQCI, sus dispositivos, sus requerimientos temporales y las necesidades que motivan la propuesta de este trabajo.
- **Capítulo 4: Diseño del sistema propuesto**. Expone la arquitectura planteada para ofrecer un servicio de sincronización homogéneo en la red, abordando aspectos como la entrega distribuida de tiempos.
- **Capítulo 5: Propuesta de entrega del servicio**. Define cómo se ofrecerá el servicio de sincronización a los usuarios o dispositivos de la red, considerando aspectos como interfaces, protocolos o modelos de servicio.

- **Capítulo 6: Integración de tecnologías de sincronización y evaluación de los resultados**. Evalúa las distintas tecnologías existentes, sus características, y su posible integración dentro del sistema propuesto.
- **Capítulo 7: Impacto del trabajo**. Destaca cómo la sincronización temporal en redes cuánticas impulsa infraestructuras críticas y contribuye a varios Objetivos de Desarrollo Sostenible
- **Capítulo 8: Conclusiones**. Resume las aportaciones del trabajo, las lecciones aprendidas y plantea posibles líneas de trabajo futuro.

Finalmente, se incluye una sección de **Bibliografía** con todas las referencias empleadas, y varios **Anexos** que recogen material complementario como el cronograma, definiciones técnicas y un glosario de términos.

Capítulo 2

Trabajo relacionado y Estado del Arte

En este capítulo se explicará el funcionamiento de los distintos mecanismos propuestos para las pruebas realizadas a lo largo del desarrollo del TFG, explicando los conceptos teóricos relacionados y las tecnologías que actualmente se están utilizando para la realización de este tipo de actividades dentro de la industria y la investigación.

2.1. Sincronización y tiempo

Aunque se puede entender que la sincronización y el tiempo son conceptos muy similares, realmente son cosas diferentes.

Leslie Lamport define el tiempo como la medida física que permite diferenciar el orden en el cual los eventos han ocurrido. Se puede decir que algo que ha ocurrido a las 13:15 ha ocurrido *antes* que algo que ha sucedido a las 13:16, igual que un evento ocurrido a las 13:16 ha ocurrido *después* que el evento de las 13:15 [5]. Esta es la unidad más extendida para la separación de eventos alrededor del mundo.

El tiempo permite ordenar la salida de los trenes de una estación para saber qué tren debe salir antes que otro. Existe, por ello, una noción de orden. Además, el tiempo puede ser anunciado de forma inequívoca, por lo que los viajeros podrán subir a uno u otro tren en función de la hora de salida de cada uno y la hora indicada en su billete. Sin embargo, en dos eventos sincronizados es irrelevante el orden en el que ocurren sus coincidencias previas o futuras. Así, los viajeros que viajen en el tren de las 9:00, si este se encuentra sincronizado con otro tren que sale de otra localización, podrán cambiar de tren en una estación intermedia sin saber qué hora es o a qué hora salió el otro tren; únicamente bajarán del tren en la estación señalada y subirán al otro tren.

Esto implica que los sistemas pueden ordenar sus procesos con una noción de tiempo, así como tener sistemas sincronizados carentes de tiempo —p. ej. relojes lógicos—, y sistemas sincronizados utilizando el tiempo como medida conjunta —p. ej. sistemas que utilizan UTC (*Coordinated Universal Time*)—.

2.2. Sincronización de tiempos en redes de ordenadores

Desde la aparición de los primeros sistemas informáticos distribuidos en los años 70 del siglo pasado, mantener una coherencia temporal entre los eventos de distintos nodos se convirtió en un reto fundamental. Cada nodo poseía su propio reloj y estos, al no estar sincronizados ni ser perfectamente estables, tendían a desviarse con el tiempo. Esta desincronización complicaba tareas como la ordenación de mensajes, la consistencia de datos o la coordinación entre procesos.

El primer enfoque formal ante este problema fue propuesto por Leslie Lamport en 1978, quien introdujo el concepto de *relojes lógicos*, una técnica que permitía ordenar eventos de forma coherente sin necesidad de sincronizar relojes físicos [5]. Más adelante, se propusieron soluciones basadas en la sincronización real del tiempo, como el algoritmo de Cristian [6] y el algoritmo de Berkeley descrito y analizado por Gusella y Zatti [7], que buscaban aproximar una hora común entre nodos mediante estimaciones del retardo y el promedio de los relojes. Estas soluciones pusieron las bases teóricas para los protocolos de sincronización modernos, aunque aún eran muy sensibles a las condiciones de red y no ofrecían precisión suficiente en todos los contextos.

2.3. Protocolos clásicos de sincronización (NTP, PTP)

Estos protocolos son ampliamente usados en redes de ordenadores modernas para la entrega de tiempos a los sistemas y, con ello, la sincronización de los programas informáticos que ejecutan.

2.3.1. Network Time Protocol

Network Time Protocol (**NTP**) [8] es un protocolo de concordancia de tiempo cuya finalidad es sincronizar los ordenadores de una red con el tiempo universal coordinado (**UTC**).

NTP se basa en una jerarquía de niveles conocida como **estratos**, los cuales determinan la calidad de la señal temporal que pueden transmitir. Se asume que el **estrato 0** está formado por fuentes de tiempo de alta precisión, como relojes atómicos o receptores GNSS (p. ej., GPS), que actúan como referencia primaria. A partir de este, cada nivel sucesivo (estrato 1, 2, etc.) se sincroniza con el anterior, de forma que cuanto mayor es el número de estrato, menor es la precisión del tiempo de referencia. El **estrato 15** representa el último nivel considerado válido, ya que por encima de este se presupone que el sistema no está sincronizado.

Los distintos dispositivos conectados a través de este protocolo se envían entre sí cuatro tramas **UDP** con la siguiente estructura:

- **T1 – Marca de envío del cliente (Originate Timestamp):** Es el instante en que el cliente envía la solicitud al servidor NTP.
- **T2 – Marca de recepción del servidor (Receive Timestamp):** Es el momento en que el servidor recibe la solicitud enviada por el cliente.
- **T3 – Marca de envío del servidor (Transmit Timestamp):** Es el instante en que el servidor responde al cliente con el paquete de respuesta.

- **T4 – Marca de recepción del cliente (Destination Timestamp):** Es el momento en que el cliente recibe la respuesta del servidor.

Con las medidas obtenidas a partir de estas marcas de tiempo, se puede calcular el ***offset*** o desfase entre el reloj del cliente y el del servidor, mediante la expresión

$$offset = \frac{(T_2 - T_1) + (T_3 - T_4)}{2},$$

la cual se utiliza para ajustar el reloj del cliente. Asimismo, se calcula el ***delay*** o retardo de ida y vuelta como

$$delay = (T_4 - T_1) - (T_3 - T_2),$$

cuya finalidad es estimar la calidad de la sincronización.

Esta tecnología proporciona una señal con una precisión media del orden de los milisegundos, aunque dicha precisión puede variar en función del estrato en el que se encuentre el cliente. Esta limitación se debe a que el sistema NTP es un programa informático que opera en la capa de aplicación, lo que introduce mayores retardos e incertidumbre en las marcas de tiempo, en comparación con protocolos que funcionan en capas inferiores.

2.3.2. Precision Time Protocol

Precision Time Protocol (**PTP**) [9] es un protocolo de sincronización temporal diseñado para lograr una alta precisión en la concordancia de tiempo entre dispositivos de una red, permitiendo la alineación con una fuente de referencia común.

PTP se organiza en una arquitectura jerárquica de tipo maestro-esclavo, en la que uno de los dispositivos de la red actúa como fuente de referencia temporal [10], y el resto ajusta su reloj en función de la información recibida. La elección de la fuente de tiempo se realiza automáticamente mediante un algoritmo denominado *best master clock algorithm* (BMCA), que compara distintas métricas de calidad temporal para seleccionar el nodo más adecuado. Una vez establecido el dispositivo de referencia, la sincronización se lleva a cabo mediante el intercambio de mensajes PTP que permiten estimar tanto el desfase temporal como el retardo en la transmisión.

Este mecanismo permite alcanzar niveles de precisión del orden de microsegundos o incluso nanosegundos, en función de la red y los dispositivos empleados.

Los dispositivos que participan en la sincronización pueden clasificarse en los siguientes tipos:

- **Grandmaster clock**: fuente principal de tiempo en la red.
- **Ordinary clock**: dispositivo que se sincroniza con otro y participa en la red PTP como maestro o esclavo.
- **Boundary clock**: actúa como intermediario entre distintos segmentos de red, sincronizándose con un maestro y proporcionando tiempo a otros nodos.
- **Transparent clock**: reenvía los mensajes PTP y corrige el retardo introducido por el dispositivo en la comunicación.

El proceso de sincronización en PTP se basa en el intercambio de **cuatro mensajes principales** entre los dispositivos participantes:

- **Sync**: enviado por el dispositivo que actúa como referencia temporal, contiene una marca de tiempo que indica el instante de salida del mensaje.
- **Follow_Up**: se utiliza cuando el reloj no puede insertar la marca de tiempo exacta directamente en el mensaje Sync. Este mensaje proporciona dicha información con mayor precisión.
- **Delay_Req**: enviado por el dispositivo que desea sincronizarse, permite medir el tiempo que tarda un mensaje en ir desde él hasta la fuente de tiempo.
- **Delay_Resp**: contiene la marca de tiempo en la que la fuente de tiempo recibió el mensaje Delay_Req.

Con las marcas de tiempo obtenidas mediante el intercambio de tramas PTP, es posible calcular el ***offset*** o desfase temporal entre el reloj del dispositivo esclavo y el del maestro. Para ello, se utilizan las marcas t_1, t_2, t_3 y t_4, que corresponden respectivamente al instante en que el maestro envía el mensaje *Sync*, el momento en que el esclavo lo recibe, el instante en que el esclavo envía el mensaje *Delay_Req* y el momento en que el maestro recibe dicho mensaje. El desfase se estima mediante la fórmula

$$offset = \frac{(t_2 - t_1) - (t_4 - t_3)}{2},$$

la cual permite ajustar el reloj del esclavo con respecto al maestro. Por otro lado, el ***delay*** o retardo de ida y vuelta se calcula como

$$delay = \frac{(t_2 - t_1) + (t_4 - t_3)}{2},$$

y representa el tiempo total que tarda un mensaje en recorrer la red en ambos sentidos, suponiendo que el camino de ida y vuelta es simétrico.

PTP es una tecnología diseñada para operar sobre la capa de enlace de datos (capa 2 del modelo OSI), lo que le permite evitar los retardos e incertidumbres asociados a las capas superiores. Gracias a esta característica, es capaz de alcanzar precisiones en el orden de los microsegundos.

2.4. Tecnologías avanzadas

Estas tecnologías permiten la generación de fuentes estables de sincronía que mejoran las prestaciones de los protocolos tradicionales como NTP o PTP. El sistema desarrollado en este trabajo presupone que podrán ser instaladas en la red MadQCI para mejorar sus capacidades.

Las fuentes de tiempo que usan los sistemas globales de navegación por satélite (GNSS) extraen el tiempo y sincronía propia de esos sistemas y las entregan al usuario. MadQCI tiene instalado un sistema SecureSync 2400 [11] capaz de entregar tanto señal de sincronía de 10 MHz como el tiempo UTC de constelaciones como GPS, Galileo o GLONASS.

Por su parte, White Rabbit (**WR**) [12] es una extensión del protocolo PTP licenciado por el CERN. Describe una topología similar a la explicada en la Sección 2.3.2, con la diferencia de que emplea un tipo específico de conmutador denominado White Rabbit Switch que, una vez conectado con otros, puede sincronizarse con ellos. Se

fundamenta también en un esquema con un sistema maestro de una red con una estructura jerárquica que se sincroniza con el resto de nodos esclavos. La sincronización será más o menos fina en función tanto de la fuente de sincronía que discipline el reloj interno del maestro (p. ej., la fuente GPS) como de las características de la red que los conecta.

Por ello, para minimizar pérdidas y mejorar la estabilidad de la señal, los dispositivos White Rabbit se conectan entre sí mediante enlaces de fibra óptica monomodo en lugar de cables de cobre. Este tipo de medio permite reducir las variaciones de retardo y las interferencias electromagnéticas, además de posibilitar la sincronización precisa entre nodos separados por distancias superiores a los 10 km, sin pérdida significativa de precisión.

Las técnicas que diferencian a WR de PTP son:

- *Synchronous Ethernet (SyncE):* Esta tecnología permite distribuir una señal de frecuencia común a todos los nodos a través del enlace físico Ethernet (nivel 2 del modelo OSI). A diferencia de PTP estándar, donde cada nodo puede utilizar su propio oscilador libre, en WR todos los nodos sincronizan su frecuencia con el maestro mediante la señal de reloj embebida en el enlace. Esto elimina el *jitter* de frecuencia y permite que todos los relojes del sistema funcionen al unísono, lo que es fundamental para la estabilidad a largo plazo.
- *Digital Dual Mixer Time Difference (DDMTD):* Este método permite medir con altísima precisión la diferencia de fase entre el reloj local de un nodo y el reloj recibido del maestro. Utiliza técnicas de mezcla de señales a frecuencias muy cercanas para amplificar las diferencias de fase y hacerlas observables con resoluciones del orden de los picosegundos. Esta medición precisa es usada para ajustar continuamente la fase del reloj local, logrando una alineación temporal extremadamente fina entre todos los nodos.

La entrega del servicio de sincronía que proveen estos dispositivos puede ser mediante la exposición de servidores PTP que actúen como maestros de otros, así como la entrega directa de señales de reloj a sistemas de alta precisión fundamentados en FPGA u otro tipo de electrónica. En este trabajo, se usa la primera forma de entrega.

Capítulo 3

Análisis del sistema MadQCI y extracción de requisitos

El capítulo de desarrollo se centrará en describir la estructura de la red MadQCI, la infraestructura sobre la que se debe instalar el sistema desarrollado, con los diferentes nodos que la componen y las necesidades de los mismos en cuanto a tiempo y sincronía.

La red MadQCI es un ensayo de red cuántica basado en diferentes dispositivos QKD, distribuidos a lo largo de 25 nodos, que está siendo puesta en marcha en la Comunidad de Madrid. Una red de este tipo es un sistema complejo, que cuenta con sistemas de infraestructura para sostener sus operaciones, como redes ópticas y de datos y sistemas informáticos; los equipos de comunicaciones cuánticas; y los sistemas de operación y control de la red, entre los que destaca el sistema de gestión de clave. Todos estos sistemas son susceptibles de precisar sincronía y tiempos externos.

3.1. Dispositivos involucrados y sus necesidades de sincronización

A continuación se explicarán los tipos de dispositivos que contiene la red y sus necesidades específicas de sincronía.

3.1.1. Dispositivos de distribución cuántica de clave

Los sistemas más importantes dentro de la red MadQCI son los dispositivos QKD. Estos dispositivos se encargan de transmitir señales cuánticas, habitualmente qubits o pares entrelazados, entre un emisor (*Alice*) y un receptor (*Bob*). Requieren una **alta** sincronización para su correcto funcionamiento, puesto que *Bob* está a la espera de las señales de *Alice* y tiene que saber 1) el momento exacto para poder medirlas o 2) la señal de sincronía con la que demodularlas.

En general, cada uno de estos sistemas utiliza un método propio de sincronización. La mayoría de estos dispositivos se sincronizan entre sí usando osciladores internos que se sincronizan mediante señales clásicas, tanto pulsos láser como datagramas. Dependiendo del dispositivo y del método de generación de clave, el sistema temporal es diferente.

En los sistemas QKD de variables discretas, que son los más habituales y antiguos, se genera y detecta la señal con una serie de componentes ópticos que generan y miden los símbolos del protocolo cuántico que estén ejecutando (p. ej., BB84 [13]). En el detector de señal pueden existir dos posibles eventos: o se detecta señal o no se detecta señal. En función del símbolo generado y de la base de medida seleccionada, estas detecciones o no detecciones tienen uno u otro significado.

Lo relevante para este trabajo es que la detección en estos sistemas ocurre en una ventana temporal que debe sincronizarse. Los sistemas de sincronía como los descritos en el capítulo anterior podrían funcionar como una fuente externa de esta sincronía.

Por otro lado, en los sistemas QKD de variables continuas, los componentes de detección de la señal cuántica usan detección coherente. Esta detección es la habitual en los sistemas de telecomunicaciones, como la radio de frecuencia modulada o las comunicaciones móviles y, por ello, se prevé que estos sistemas tengan un recorrido comercial mayor.

Para su funcionamiento, los dos sistemas deben estar sincronizados con la misma fase. Por ello, se emite un pulso láser junto con la señal cuántica y se detecta su fase en el receptor (p. ej., usando un *phase-locked loop*). Así, el modulador y demodulador de los dos sistemas usarán una fuente de señal sincronizada.

Todos los dispositivos QKD son ajenos a la red de sincronía desarrollada en este trabajo, puesto que su integración con fuentes externas de sincronía no está en el *roadmap* de la industria, pero sí que tienen interés en otras funcionalidades de los sistemas QKD. Es el caso de los KMS internos en cada QKD, que podrían usar la fuente de tiempo externa para mejorar su seguridad, su integración con los KMS de la red, etc.

3.1.2. Otros servicios cuánticos

La red otorga servicios cuánticos más allá de la distribución cuántica de clave. Estos servicios por lo general están exentos de necesitar una señal de sincronía, puesto que se consultan en momentos específicos haciendo que la complejidad se transporte a la máquina que realiza dicha consulta.

Estos servicios pueden ser los generadores cuánticos de números aleatorios (QRNG por sus siglas en inglés), que pueden ser útiles en el campo de la investigación para cualquier persona que necesite una cadena de aleatoriedad real dentro de su experimento. Por ejemplo, un investigador de *front end* puede estar interesado en hacer peticiones a este tipo de interfaces de programación de aplicaciones para generar contraseñas de carácter aleatorio (que no seguras) para sus diferentes usuarios o un físico podría utilizarlo para investigar las desigualdades de Bell [14].

Estos sistemas funcionan a través de eventos aleatorios intrínsecos a la física cuántica, p. ej., pasar un fotón por un divisor de haz. Una vez realizado este evento, se traduce a un binario, generando así un número con una aleatoriedad plena.

La red MadQCI integra varios sistemas QRNG, tanto del fabricante ID Quantique como de Quside. Los modelos que se han instalado se fundamentan tanto en interfaces USB como en PCIe, así como sistemas tipo servidor que integran el QRNG para prestar su servicio de forma remota. Todos ellos se pueden beneficiar de una sincronía

mejor en la red.

3.1.3. Dispositivos clásicos

Los distintos sistemas de infraestructura que sustentan las operaciones de la red cuántica son susceptibles de necesitar fuentes de tiempo y sincronía externas.

Es el caso de los sistemas informáticos que alojan las distintas máquinas virtuales con los servicios que sustentan las operaciones y aplicaciones de la red. Cada uno de los nodos aloja máquinas que corren sobre Proxmox, un hipervisor capaz de generar esas máquinas virtuales con el fin de que los posibles usuarios de las aplicaciones puedan acceder a los diferentes servicios. Estas máquinas están diseñadas para que investigadores de todo tipo sean capaces de realizar los diferentes experimentos sin tener conocimientos avanzados dentro del ámbito de la informática.

Todos estos nodos deben tener un nivel de sincronía común, puesto que son los que se van a encargar de acceder a los diferentes recursos de manera concurrente, además de mantener los experimentos que van a funcionar en las mismas. También necesitarán acceso a estos tiempos para operaciones seguras, como puede ser generar certificados digitales para la autenticación.

3.2. Suposiciones y requisitos del diseño

Tomando como fundamento el diseño de MadQCI, en los siguientes apartados se explicarán las suposiciones de las que parte la red de distribución y entrega de tiempos, así como sus requisitos para ser funcional.

3.2.1. Suposiciones

En la tabla 3.1 se listan las suposiciones que la red cumple y que son parte necesaria para poder generar un sistema de sincronía correcto. Estas suposiciones son válidas para todas las tecnologías propuestas.

3.2.2. Requisitos

A continuación se listarán los requisitos funcionales y no funcionales del sistema. Estos requisitos se deben cumplir para un correcto funcionamiento de la red de distribución y entrega de tiempos.

3.2.2.1. Requisitos funcionales

Los requisitos funcionales son las especificaciones detalladas que describen las acciones y comportamientos que un sistema debe realizar para cumplir con las necesidades y expectativas de los usuarios. En este caso específico hacen referencia a las garantías que otorga la red a los diferentes dispositivos en la tabla 3.2.

3.2.2.2. Requisitos no funcionales

Los requisitos no funcionales son criterios que determinan el comportamiento del sistema de sincronización bajo ciertas condiciones, sin describir funciones específicas. En la tabla 3.3 se citan los aplicados en este trabajo.

ID	Suposición	Justificación
A1	Red LAN de alto rendimiento	La baja latencia y el control total sobre la red lo hacen viable.
A2	Soporte de múltiples protocolos	Todos los nodos soportan todos los protocolos propuestos (NTP, PTP, WR).
A3	Existen dispositivos intermedios que pueden actuar como nodos intermedios	La señal se podrá gestionar a través de máquinas intermedias para no sobrecargar la red.
A4	Enlaces punto a punto *full-duplex*	Se asegura latencia determinista y simetría de propagación.
A5	Fuente de tiempo externa confiable	Existe una fuente de tiempos fiable que transmite la conexión a toda la red.
A6	La topología de la red es desconocida	La topología de la red es desconocida y variable; las instancias de las máquinas no son fijas.
A7	Todos los dispositivos tienen visibilidad entre sí	La topología de la red garantiza que todos sus dispositivos son capaces de conectarse entre sí.

Cuadro 3.1: Suposiciones del sistema de distribución y entrega de tiempos para red MadQCI.

ID	Requisito	Descripción
RF1	Soporte de NTP, PTP y WR	El sistema debe permitir sincronización y entrega de tiempos mediante cualquiera de estos protocolos.
RF2	Selección automática de protocolo	En cada nodo, se usará el mejor protocolo según precisión requerida.
RF3	Coherencia temporal entre nodos	La diferencia temporal debe estar por debajo del umbral definido.
RF4	Supervisión de sincronía	Se debe poder diagnosticar el estado de sincronización en cada nodo.
RF5	Entrega del servicio	En cada nodo, se deben proveer los puntos de acceso necesarios para que los usuarios sincronicen sus propios relojes.

Cuadro 3.2: Requisitos funcionales del sistema de sincronización

ID	Requisito	Descripción
RNF1	Precisión según protocolo	WR: < 100 ns, PTP: < 1 µs, NTP: < 10 ms.
RNF2	Alta disponibilidad	Disponibilidad mínima anual del 99.99 %.
RNF3	Baja latencia de sincronización	RTT de los mensajes < 1 ms, excepto para NTP.
RNF4	Compatibilidad IEEE 1588-2008	Todos los nodos deben soportar, al menos, PTPv2.
RNF5	Autoconfiguración topológica	La red debe adaptarse automáticamente a la topología de la red y sus cambios, lo que incluye las fuentes externas de sincronía y la prestación del servicio.
RNF6	Virtualización	La red debe desplegarse en máquinas virtuales o mediante contenedores Linux.

Cuadro 3.3: Requisitos no funcionales del sistema de sincronización

Capítulo 4

Diseño del sistema propuesto

En este capítulo se explicarán las diferentes propuestas para garantizar la funcionalidad plena de la red de sincronía. Se adjuntarán esquemas básicos reducidos y, después, el diseño propuesto para la red completa.

Es relevante señalar que todos los servicios de la red están diseñados para funcionar dentro de máquinas virtuales a través del hipervisor Proxmox VE. Para que la actualización de esta hora sea constante, se aplicará el uso de herramientas específicas como Chrony [15] o como Ptp4linux [16]. A continuación se explicarán los distintos tipos de redes que se pueden realizar dependiendo de la tecnología utilizada y, por último, se mostrará una posible arquitectura de red.

4.1. Arquitectura general del sistema

A continuación se explicarán distintas posibilidades de gestión de red en función de las tecnologías existentes. Dependiendo de la tecnología utilizada, previamente explicada en la Sección 2, la arquitectura podrá ser una u otra. En todas las tecnologías se explicará un breve ejemplo de cómo sería un sistema mono-nodo y, más adelante, se presentarán varios esquemas para posibles tipos de red, dependiendo de la conectividad de la misma.

Como se explica en la Sección 3, se toma la suposición A5 de la Tabla 3.1, asumiendo que siempre hay acceso a una fuente de tiempo externa y confiable.

4.1.1. Introducción al sistema de máquinas virtuales en el hipervisor Proxmox

Antes de detallar las distintas formas de prestación del servicio de sincronización, es importante introducir brevemente el entorno virtualizado sobre el que se despliega la red. Para ello se ha utilizado Proxmox VE, un hipervisor de tipo 1 basado en tecnologías como KVM [17] y QEMU [18], que permite ejecutar múltiples máquinas virtuales sobre el mismo *hardware* físico de forma eficiente y aislada.

Cada nodo de la red se ejecuta como una máquina virtual (VM). Estas VMs disponen de su propio sistema operativo y entorno de red, pero comparten los recursos físicos del servidor anfitrión. A través de Proxmox, es posible asignar a cada VM una cantidad concreta de CPU, memoria RAM, almacenamiento y adaptadores de red virtuales.

El sistema se encarga de repartir estos recursos de forma dinámica, garantizando que cada instancia pueda funcionar de manera autónoma sin interferencias directas con el resto.

Esta arquitectura facilita la creación de un entorno de pruebas flexible y reproducible, permitiendo desplegar múltiples topologías sin necesidad de contar con varios equipos físicos. Además, al centralizar la gestión mediante la interfaz web de Proxmox, se simplifica el control y la monitorización del rendimiento de cada nodo virtual.

4.1.2. Ejemplo de arquitectura NTP

En el caso de NTP, el *stratum* 1 sirve para definir la fuente fiable y estable de tiempos. Todos los demás *stratum* dependen de cómo se realice el cableado entre las distintas máquinas; es decir, las máquinas que tengan acceso al *stratum* 1 serán las que, por definición, se conviertan en *stratum* 2 y, así, sucesivamente.

Nótese que esto es válido tanto para cableado de red (p. ej., Ethernet) como para la conectividad interna que se da en los sistemas informáticos entre hipervisor y máquinas virtuales, o entre estas últimas. Por ello, dado que la red de distribución y entrega de tiempos estará virtualizada, los usuarios del sistema serán de un *stratum* 3 como mínimo.

Para que, en caso de caída de alguno de los servicios, los servicios dependientes se puedan autoconfigurar, tiene sentido que, dentro de la red, cada dispositivo del *stratum* X sea consciente de la existencia de los dispositivos desde su *stratum* hasta el *stratum* 2. De esta manera, podrá ajustarse al sistema que sea capaz de proporcionar la fuente de tiempo más fiable sin sobrecargar la red.

En la figura 4.1 se muestra un ejemplo de una configuración NTP dentro de un nodo de MadQCI. En ella se pueden observar hasta 3 *stratum*, cada uno con visibilidad del *stratum* superior y de los dispositivos del mismo rango.

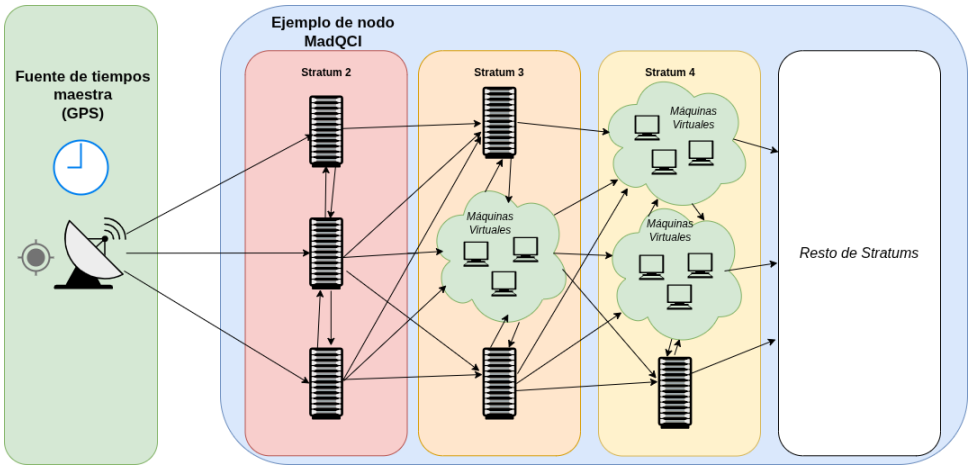

Figura 4.1: Ejemplo de configuración NTP para un solo nodo de la red MadQCI. La fuente de tiempos externa, en realidad, es el *stratum* 1 en NTP, por lo que cada uno de los sucesivos niveles en el cableado (tanto Ethernet como intra-hipervisor) son el *stratum* 2, 3...

Puesto que la red MadQCI tendrá una topología variable, se pretende realizar una

pequeña interfaz *software* para, de esta manera, poder actualizarla de manera automática la topología dentro de todos los sistemas de la red.

Para asegurar el correcto funcionamiento de NTP, la recomendación es que todas las redes sigan una estructura jerárquica [19], para asegurar su estabilidad y escalabilidad. En la figura 4.2 se puede observar cómo sería una red formada por varios nodos NTP. En este dibujo, se asume que todos los aparatos de un mismo *stratum* tienen visibilidad entre sí y, además, visibilidad de todos los dispositivos de los *stratum* inferiores.

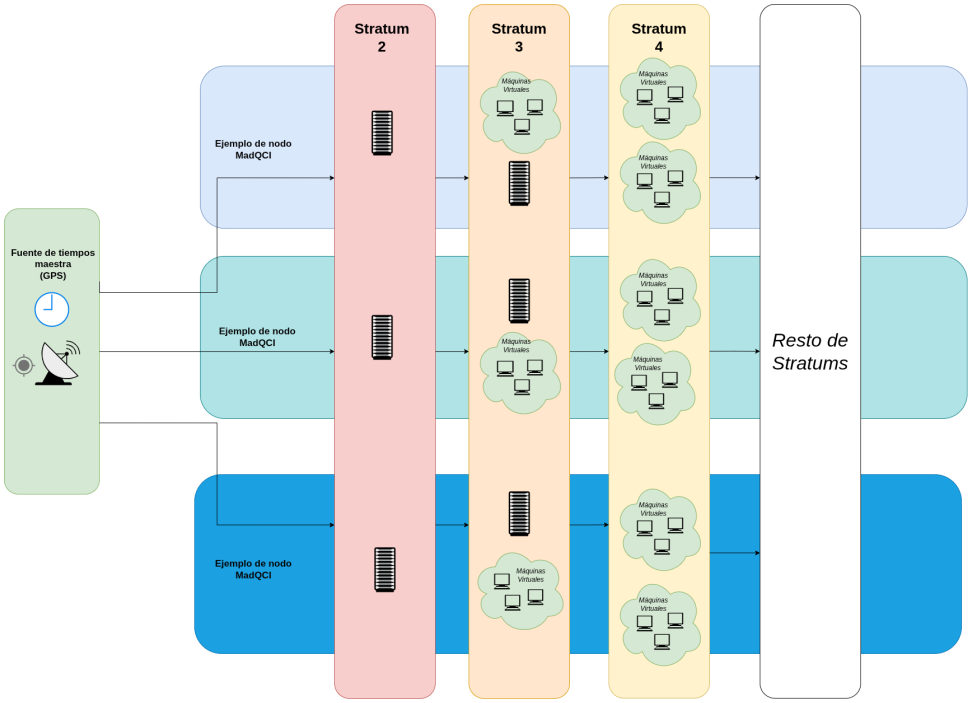

Figura 4.2: Ejemplo de la configuración NTP en un escenario de la red MadQCI con varios nodos. En función de las características de cada nodo, la estratificación puede ser más o menos profunda en cada uno.

4.1.3. Ejemplo de arquitectura PTP

En el caso de PTP, todos los nodos que emiten paquetes envían a los dispositivos de la red un mensaje *Announce* [16]. Este mensaje permite que, de forma automática, la máquina que desea sincronizarse pueda conocer cuáles son las fuentes de tiempo disponibles.

Los mensajes *Announce* incluyen los siguientes campos:

- **priority1:** Configurado manualmente; distingue roles (por ejemplo, GPS preferido).
- **clockClass:** Tipo de fuente de tiempo (GPS, rubidio, reloj interno, etc.).
- **clockAccuracy:** Precisión medida o estimada del reloj.

- **offsetScaledLogVariance:** Estabilidad del reloj (variación de frecuencia).
- **priority2:** Valor de desempate adicional (menor valor es preferido).
- **clockIdentity:** Identificador único del reloj (similar a una dirección MAC).

Una vez recibe el listado de todos los relojes disponibles, la máquina ejecuta un algoritmo llamado *best master clock* (BMC) [9], evaluando los datos recibidos de los diferentes mensajes *Announce*. Supongamos que el reloj recibe dos conjuntos de datos distintos, llamados A y B; los comparará en el orden mostrado en la tabla 4.1. En caso de que el primer campo sea menor en uno de los conjuntos, el reloj se quedará con ese. Si no, pasará al siguiente campo.

Paso	Campo comparado
1	`priority1`
2	`clockClass`
3	`clockAccuracy`
4	`offsetScaledLogVariance`
5	`priority2`
6	`clockIdentity` (desempate final)

Cuadro 4.1: Orden de comparación del algoritmo BMC en IEEE 1588

Si el reloj del dispositivo es mejor, este se convierte en *grandmaster* y comienza a enviar paquetes *Announce*; si no, pasa a ser *slave* o reloj pasivo. Esto solo aplica a los *boundary clocks*, que actúan como *grandmaster* en un nodo local, y a los *ordinary clocks*, que funcionan como clientes. La red general contará con *transparent clocks*; estos terminales sirven para añadir el campo *correctionField* a los paquetes de sincronización y corregir así errores en redes con múltiples dispositivos o estructuras jerárquicas.

En la figura 4.3 se muestra un ejemplo de cómo funcionaría un sistema PTP en un nodo exclusivo. Este ejemplo carece de *transparent clocks* puesto que, al tratarse de un nodo único, no serían necesarios. El *boundary clock* actuaría como *grandmaster* del nodo de la red MadQCI. Las máquinas con el hipervisor funcionarían, a su vez, como *boundary clocks* para las máquinas virtuales que hospedan, garantizando así que la red no se sobrecargue.

Topologías físicas en redes PTP

- **Topología en estrella:** El *grandmaster* se conecta directamente a los esclavos a través de un conmutador central. Ofrece baja latencia y simplicidad, ideal para redes compactas con un solo punto de distribución. Ejemplo gráfico en la figura 4.4.
- **Topología jerárquica (árbol):** El *grandmaster* está en la raíz, con *boundary clocks* que reparten el tiempo a diferentes ramas. Es escalable y adecuada para redes grandes o distribuidas. Es uno de los esquemas que más se replicará en la red. Véase el ejemplo en la figura 4.5.
- **Topología en cadena (*daisy chain*):** Los dispositivos están conectados en serie. Es económica y útil en entornos lineales, aunque presenta mayor acumulación

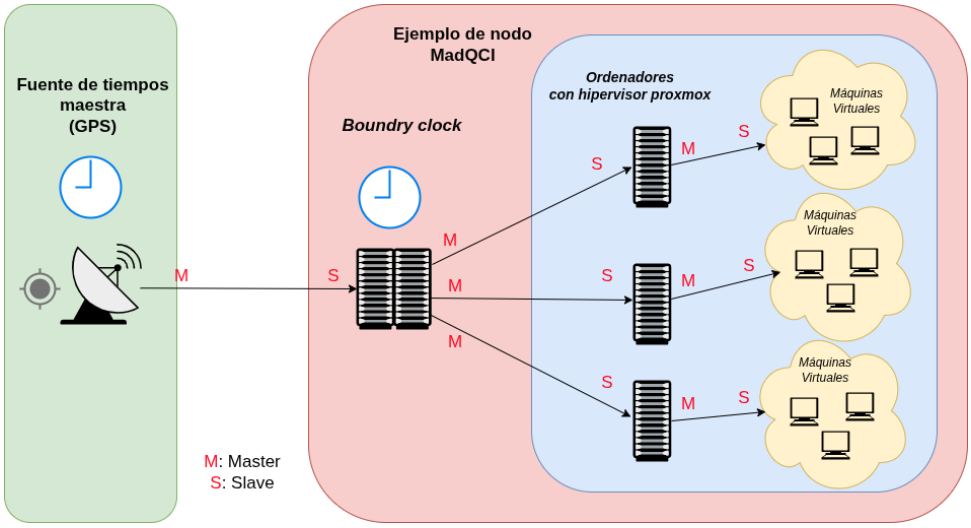

Figura 4.3: Ejemplo de configuración PTP para un solo nodo de la red MadQCI. A diferencia del NTP, no existen los estratos, sino una concatenación de maestros y esclavos determinada por los sistemas con mejor precisión.

de retardo y menor tolerancia a fallos. Solo se aplicará en nodos que estén conectados a un único nodo y que no tengan conexión directa a la fuente de tiempo. Véase el ejemplo en la figura 4.6.

- **Topología en malla (*mesh*):** Cada nodo puede conectarse a múltiples nodos. Tiene alta tolerancia a fallos, pero es compleja de gestionar. Se aplicará en nodos finales que tengan acceso a varios nodos con fuentes fiables. Véase el ejemplo en la figura 4.7.
- **Topología en anillo:** Variante de la cadena donde el último nodo se conecta al primero. Mejora la redundancia, pero requiere protocolos de control para evitar bucles. Es útil en nodos que tengan conexión con varios otros nodos. Véase el ejemplo en la figura 4.8.
- **Topología híbrida:** Combinación de varias topologías (p. ej., estrella + árbol). Flexible y adaptable a redes con distintos requisitos físicos o funcionales. Esta será la topología que se aplicará en todo el sistema, combinando todas las anteriores. En la sección 4.2 se explicará un prototipo de red para una posible tipología.

4.1.4. Ejemplo de arquitectura WR

White Rabbit es un protocolo basado en **PTPv2** [10]; por tanto, los diseños de las redes **WR** son equivalentes a los explicados en la sección 4.1.3, pero requieren el uso de *hardware* específico.

Para poder utilizar este tipo de sistemas, todos los nodos deben disponer de un conmutador de red específico que gestione el protocolo, por ejemplo, el White Rabbit Switch – Low Jitter [20]. Esto implica que los *boundary clocks* de cada nodo se sustituyen por *transparent clocks* implementados mediante dicho *hardware*, dando lugar a una arquitectura como la mostrada en la figura 4.9.

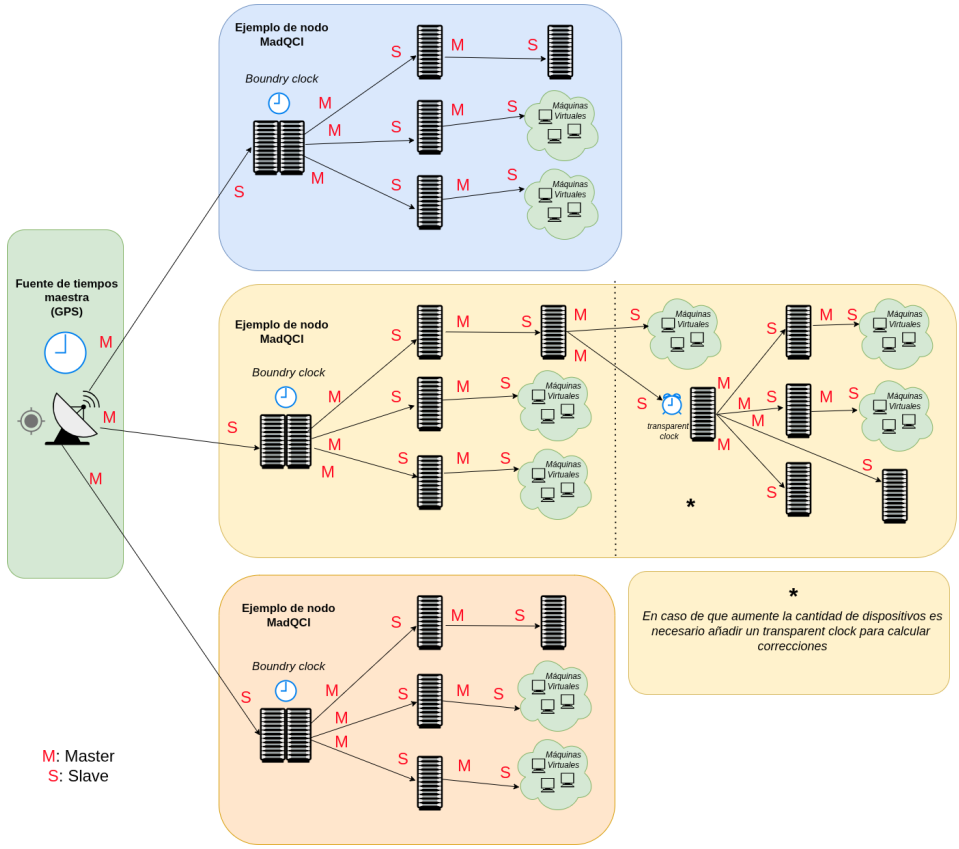

Figura 4.4: Ejemplo de configuración PTP en estrella para tres nodos de la red MadQCI. Se puede observar que todos los nodos están directamente conectados a la fuente de tiempos, lo que permite recibir la sincronización de forma precisa y sin depender de saltos intermedios. Dentro de cada nodo, las máquinas virtuales reciben el tiempo a través de *clocks* intermedios, permitiendo un reparto jerárquico y escalable de la sincronización.

4.2. Propuesta de sistema para la actual propuesta de la red MadQCI

Como se menciona previamente en la tabla 3.1, la estructura de la red es desconocida, puesto que todavía no está implementada. En el artículo [4] se propone una estructura como la mostrada en la figura 4.10, que muestra una estructura mixta con distintas capacidades.

- **Topología en estrella:** La estructura central de la red se basa en una topología en estrella, donde el nodo **UPM Rectorado** actúa como *grandmaster*, dado que está previsto que allí se reciba un patrón de frecuencia ultraestable por una colaboración con el Centro Español de Metrología (CEM). Este se conecta directamente con múltiples nodos periféricos como ETSIT, ETSIINF, CETSE, CEM, INTATO, MadQCI TEF, Cedint, UCM y UC3M. Esta disposición permite una distribución de tiempo con baja latencia y gran simplicidad en la gestión de la sincronización.
- **Topología jerárquica (árbol):** A partir del nodo central, ciertos nodos intermedios actúan como *transparent clocks* para distribuir la señal de sincronización a

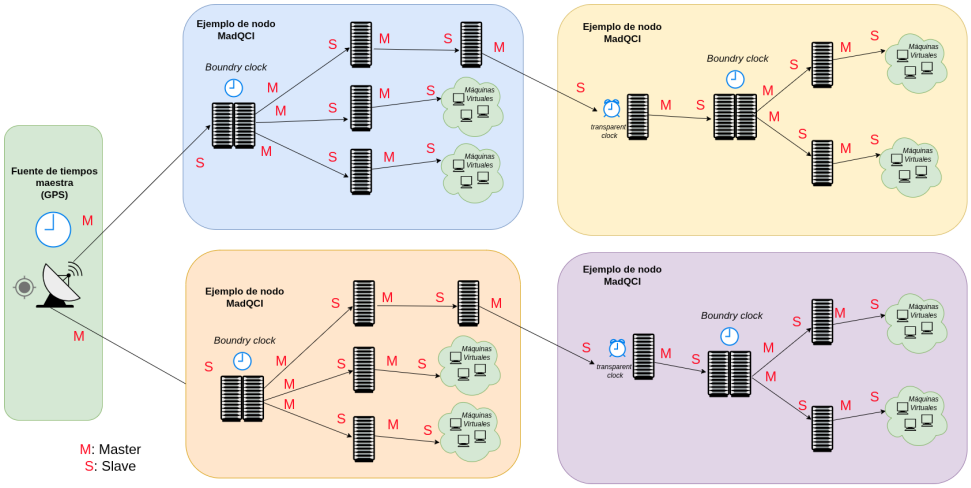

Figura 4.5: Ejemplo de configuración PTP en árbol para la red MadQCI. La sincronización se distribuye jerárquicamente desde la fuente de tiempo GPS a través de múltiples nodos conectados mediante relaciones *Master/Slave*, utilizando *Boundary Clocks* y *Transparent Clocks* para minimizar el error acumulado y mantener la precisión temporal entre máquinas virtuales.

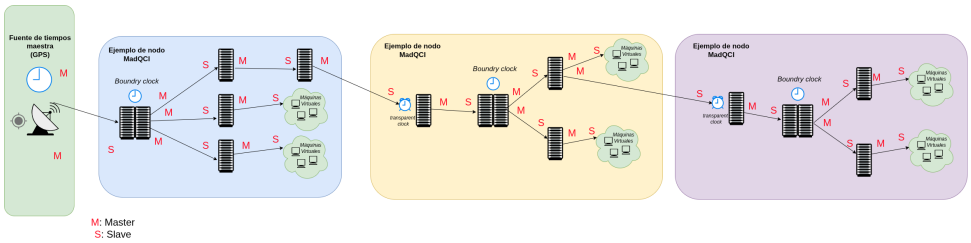

Figura 4.6: Ejemplo de configuración PTP en cadena (*daisy chain*) para la red MadQCI. La sincronización se propaga secuencialmente entre nodos mediante *Boundary Clocks* y *Transparent Clocks*, permitiendo extender la red en línea sin perder precisión temporal en las máquinas virtuales.

otros nodos secundarios. Ejemplos notables incluyen:

- UCM, que redistribuye hacia UAM y CSIC.
- UC3M, que conecta con URJC, IM-NW e IM-SW.
- CAIT, que agrupa varios nodos internos en su dominio.

Este modelo permite escalar la red de forma organizada, reduciendo la carga sobre el nodo maestro.

- **Topología en cadena (*daisy chain*):** Se identifica en segmentos donde los nodos están conectados secuencialmente, como la cadena formada por ISCIII y CEM, o la que conecta IM-SW con RM. Esta solución es útil en ubicaciones lineales o nodos aislados que no requieren redundancia directa, aunque introduce mayor retardo acumulado.
- **Topología en anillo:** En los agrupamientos locales como MadQCI TEF y CAIT,

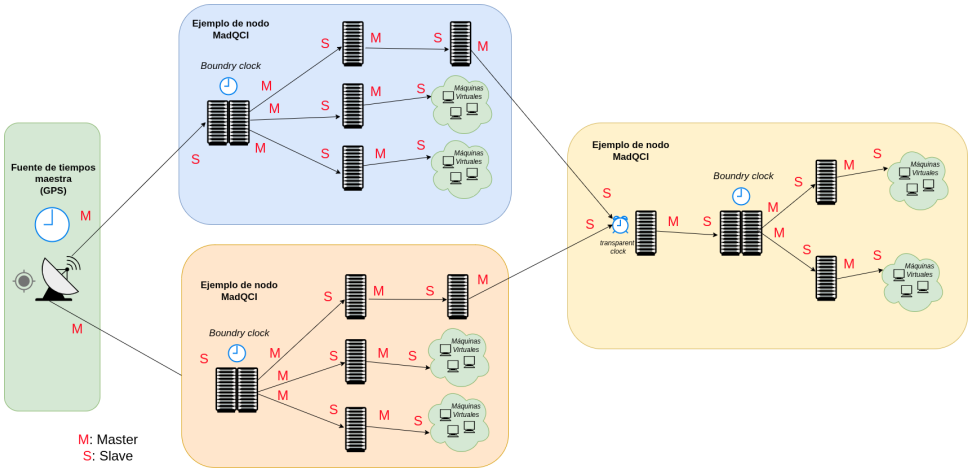

Figura 4.7: Ejemplo de configuración PTP en malla para la red MadQCI. Los nodos están interconectados mediante múltiples enlaces, combinando *Boundary Clocks* y *Transparent Clocks* para mejorar la resiliencia y distribuir la sincronización temporal de forma redundante entre máquinas virtuales.

se infiere una estructura en anillo que conecta los nodos del círculo entre sí. Esto permite mantener la redundancia en el acceso a la sincronización temporal dentro de un dominio físico limitado.

- **Topología en malla (*mesh*):** En los entornos con alta interconectividad, como la zona que incluye a CSIC, UC3M, UCM y sus conexiones cruzadas, se emplea una topología en malla que garantiza múltiples rutas de sincronización. Esta solución es fundamental para asegurar tolerancia a fallos y robustez en los nodos críticos.

Esquema PTP-White Rabbit propuesto para todos los nodos de la red MadQCI

En la figura 4.11 se puede observar una topología válida tanto para PTP como para White Rabbit con todas las conexiones de la red.

La dirección de la flecha indica desde dónde parte la señal maestra y hacia dónde se dirige. Esta red está separada en diferentes colores para poder observar qué tipología se ha utilizado en cada nodo.

- **Topología en estrella:** Los nodos CETSE, CEM, ISCIII, ETSIT e INTAMA están todos conectados al nodo de Rectorado, que actúa como *master clock* de la red. Estos nodos tienen garantizada la mejor conexión posible al tiempo de la red.
- **Topología en anillo Telefónica:** Los nodos MadQCI Telefónica 1, MadQCI Telefónica 2, MadQCI Telefónica 3 y MadQCI Telefónica 4 tienen todos conexión entre sí para garantizar una mejor entrega del servicio. El nodo **MadQCI Telefónica 1** es el que tiene mejor conexión de todos, debido a que está conectado con el de Rectorado.
- **Topología en anillo UPM:** Los nodos UPM Cedint, RM-IM-SW, CAIT y ETSINF

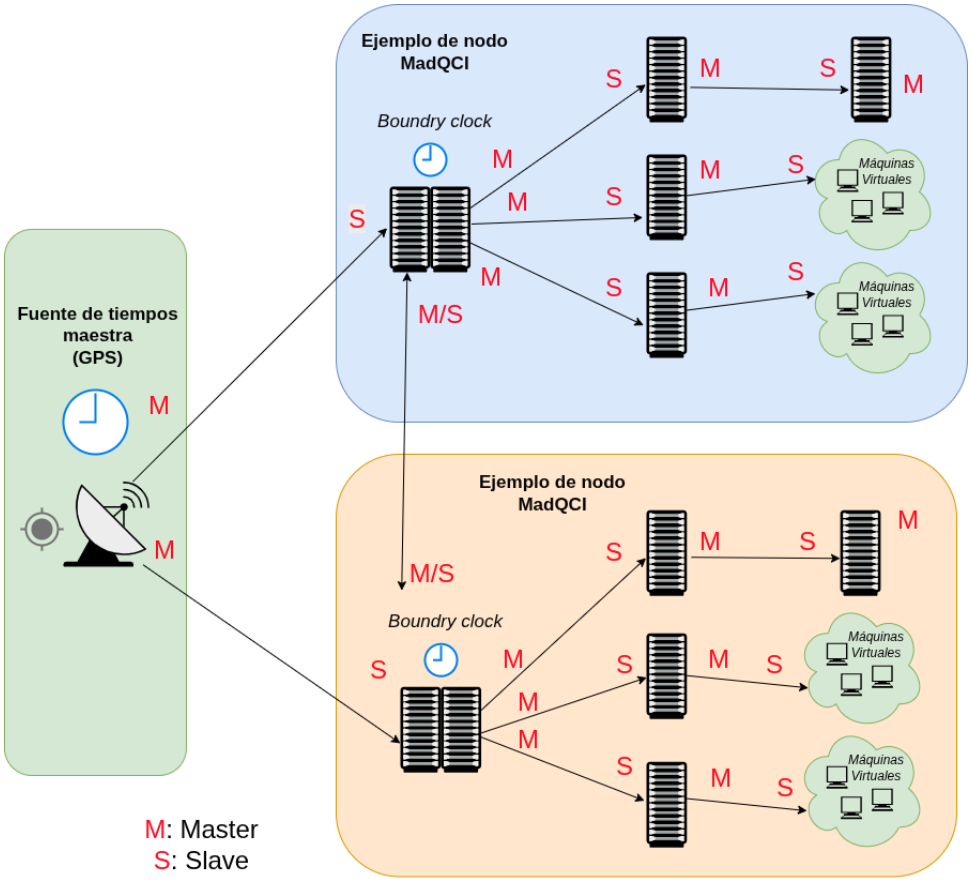

Figura 4.8: Ejemplo de configuración PTP en anillo para la red MadQCI. Cada nodo puede actuar simultáneamente como *Master* y *Slave* (M/S), lo que permite mantener la sincronización incluso en caso de fallo de uno de los enlaces, aumentando la tolerancia a fallos del sistema.

también están conectados en forma de anillo, siendo el nodo Cedint el que mejor fuente de tiempos tiene al estar conectado a Rectorado.

- **Topología en árbol:** Los nodos UCM, CSIC, UC3M y UAM describen una topología en árbol. El nodo **UCM** recibe la conexión de la fuente temporal de Rectorado y la distribuye entre UAM, UC3M y CSIC. Después, CSIC la distribuye a su vez al nodo UAH.
- **Topología mixta:** Podemos apreciar que este nodo en sí tiene tanto topología de árbol como topología en cadena. La topología de cadena empieza desde el nodo UCM hasta el nodo UC3M, pasando primero por otro nodo UC3M y por IM-NW, generando una topología de 3 nodos en cadena. La estructura de árbol se aprecia cuando el nodo IM-MW le pasa su marca temporal también al nodo URJC, creando así la estructura mixta.
- **Topología en cadena:** El nodo de la UNED, al estar aislado, depende de una topología en cadena, recibiendo su marca temporal a través de RM-IM-SW, nodo conectado a la topología en anillo de la UPM.

El protocolo PTP, al ser jerárquico, se generaría desde varios nodos muy similares

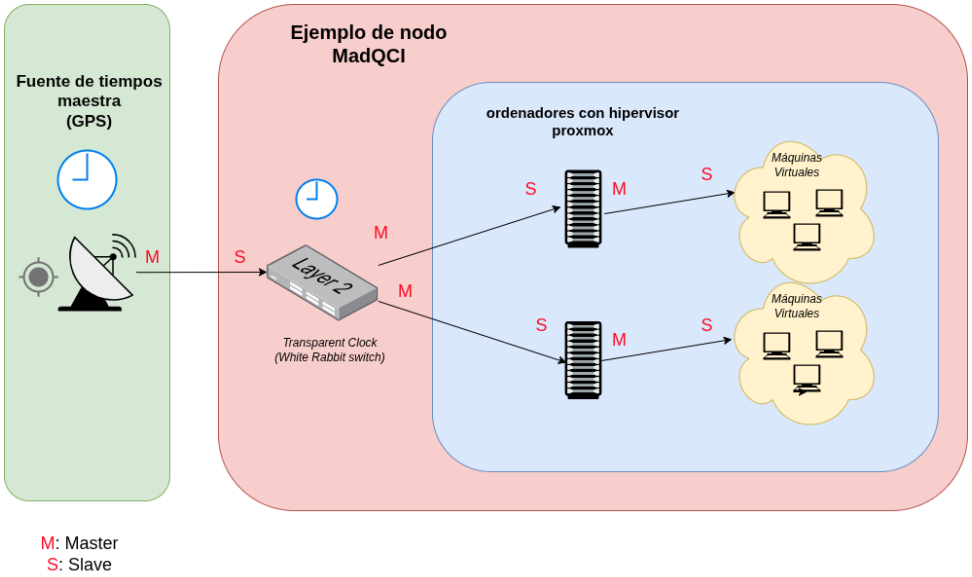

Figura 4.9: Ejemplo de configuración WR mono-nodo. Nótese que la conexión entre la fuente de tiempos GPS y el conmutador White Rabbit se realiza mediante una señal de sincronía de 10 MHz, mientras que el resto de dispositivos utilizan White Rabbit o PTP.

a los de la figura 4.2, uno diferente por cada una de las nubes representadas en la figura 4.11.

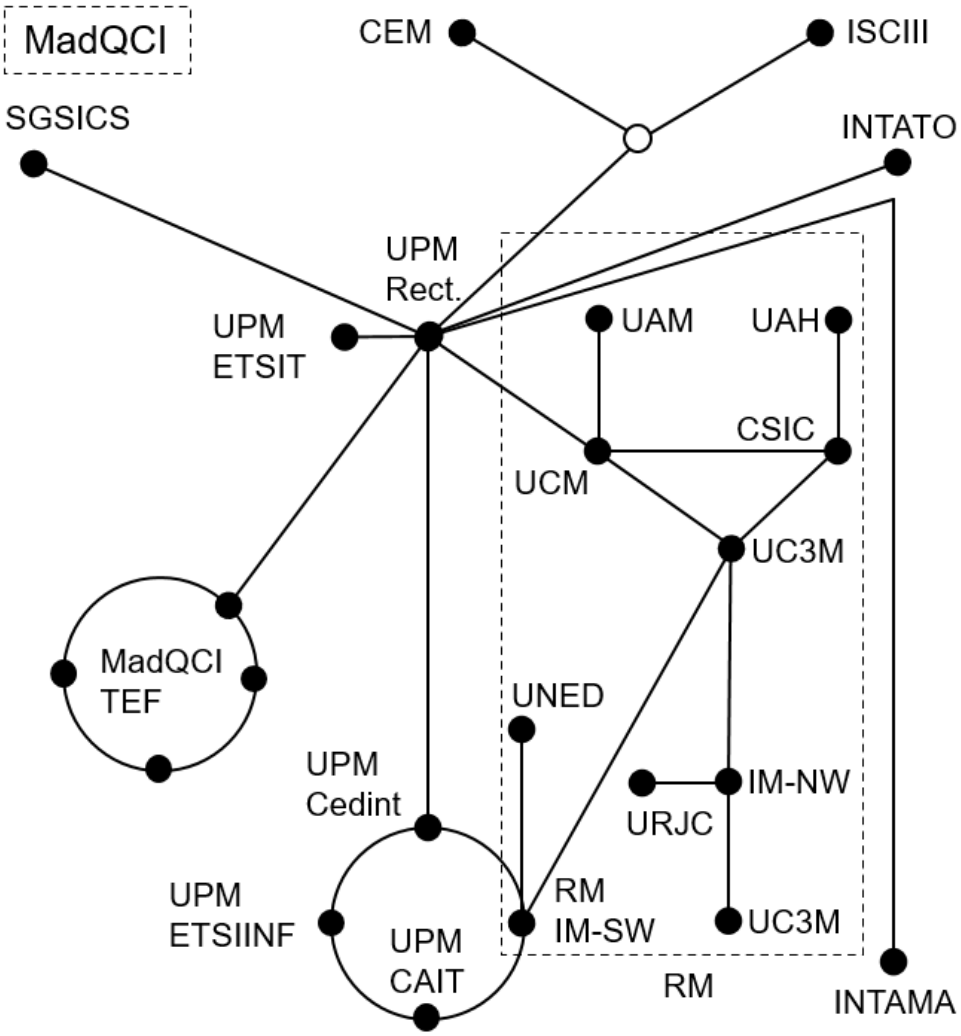

Figura 4.10: Topología de la red MadQCI tal y como estaba a mediados de 2025. Esta estructura se ha usado como base para extraer los requisitos de diseño del sistema de distribución y entrega de tiempos.

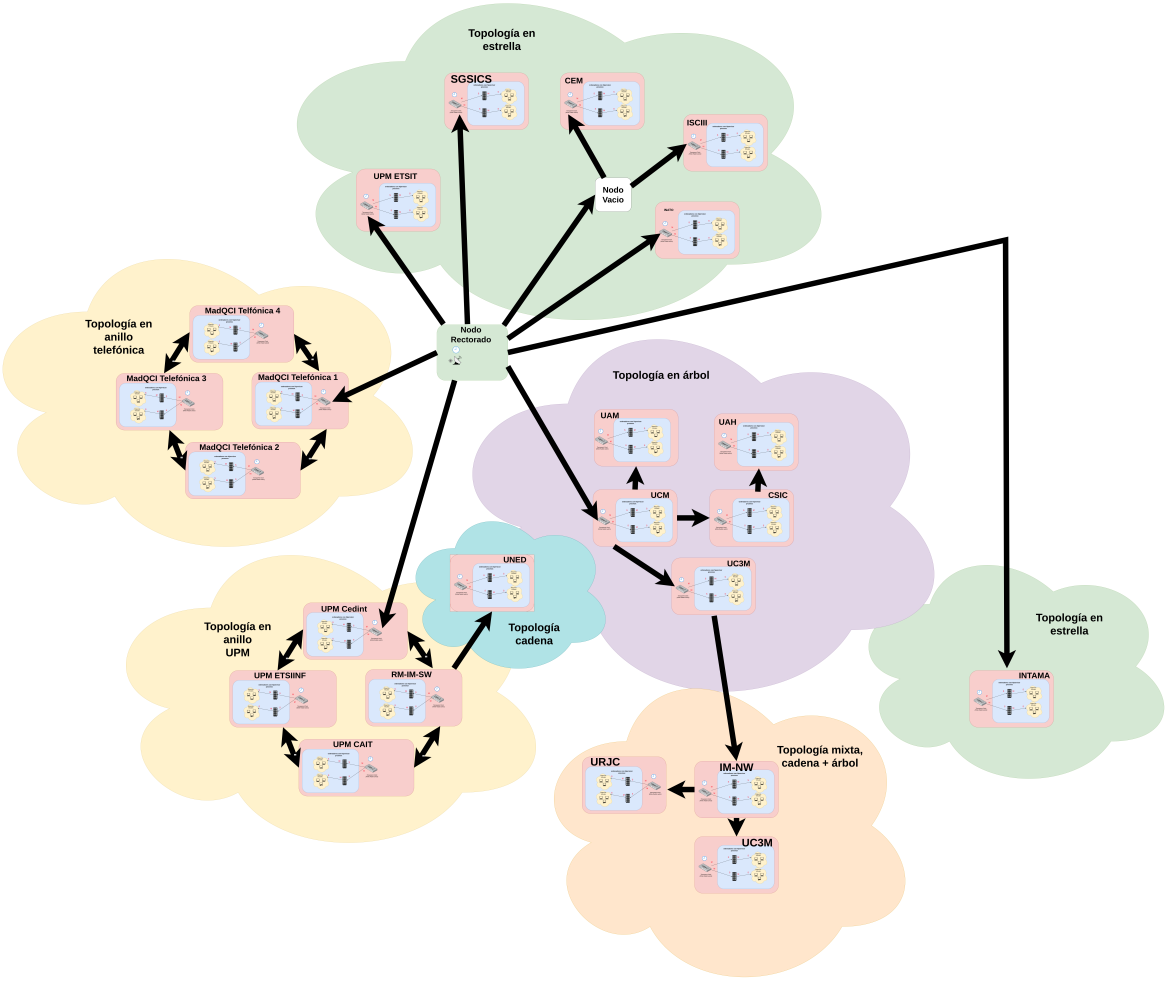

Figura 4.11: Esquema propuesto PTP/WR, cada una de las nubes representa el tipo de topología que se aplica para los nodos que están dentro.

Capítulo 5

Propuesta de entrega del servicio

En esta sección, se explicarán posibles entregas a usuarios para poder gestionar esta red de tiempos de manera segura. Esta entrega del servicio puede ser útil para posibles acciones de auditoría, generando logs para verificar de manera segura cuándo han sucedido posibles acciones dentro de la red —p. ej., cuando un usuario ha realizado una petición de clave a un dispositivo—.

5.1. Interfaces para la provisión de tiempo (API REST, XaaS)

La red de sincronía no está pensada únicamente para suministrar una señal de reloj correcta a todos los dispositivos de la red, sino también para ofrecer un servicio a los usuarios, permitiéndoles verificar acciones dentro del sistema. Diferentes modelos de servicio pueden implementarse en máquinas virtuales dentro del servidor Proxmox.

Para posibles investigaciones que requieran servicios de sincronía, se ha pensado en suministrar distintos tipos de interfaces que permitan acceder a dicho servicio.

A continuación, se explican distintos métodos que pueden utilizarse para la entrega del servicio:

- **API REST:** Exponer un servicio REST que permita consultar la hora sincronizada desde cualquier punto de la red, facilitando su integración en otras aplicaciones o sistemas.
- ***Infrastructure as a Service* (IaaS):** Permitir la creación de máquinas virtuales dentro del entorno Proxmox ya configuradas con servicios de sincronización temporal, listas para su uso en proyectos de investigación o entornos de prueba. Para ello, se crearían distintos *templates* diseñados para integrarse dentro de la red de sincronía con las demás máquinas.
- ***Platform as a Service* (PaaS):** Proporcionar una interfaz o librería en `/dev` que permita a los desarrolladores construir aplicaciones que consuman el tiempo sincronizado como si fuera un recurso del sistema.
- ***Software as a Service* (SaaS):** Ofrecer herramientas listas para usar, como un programa en `/bin/time`, que devuelva la hora sincronizada sin necesidad de configuración adicional por parte del usuario, permitiendo así tomar registros como llamadas al sistema.

5.2. Formatos y protocolos de entrega a usuarios

A continuación, se explican los distintos métodos de entrega de un sistema de tiempos en función de las necesidades de la interfaz descrita en 5.1. El tiempo se entregará siempre en formato **UTC**. La forma en que se proporcione este tiempo dependerá de la interfaz que lo consuma; por ejemplo, la API REST utilizará formato JSON, mientras que la interfaz *Infrastructure as a Service* devolverá paquetes temporales idóneos para la máquina virtual creada.

5.2.1. API REST

En esta sección se propone una configuración básica de cómo funcionaría un sistema que ejecuta una API REST para garantizar la entrega de tiempos sincronizados a los distintos sistemas dentro de la red.

1. `GET /api/time/now`

Descripción	Devuelve la hora sincronizada actual, firmada digitalmente.
Método	GET
Respuesta ejemplo	`{ "timestamp": "2025-05-29T15:03:21.123456Z", "source": "grandmaster", "precision_ns": 100, "signed": true, "signature": "MEUCIQC7e9S..." }`

2. `POST /api/time/stamp`

Descripción	Registra un evento desde un nodo y devuelve su marca de tiempo firmada.
Método	POST
Cuerpo de solicitud	`{ "node_id": "node-01", "event": "sensor_triggered", "meta": { "sensor_id": "s7", "voltage": "2.34" } }`
Respuesta ejemplo	`{ "event_id": "evt-8f21c9", "timestamp": "2025-05-29T15:05:33.778Z", "signature": "MEUCIGn8P9...==" }`

3. `GET /api/time/status`

Descripción	Consulta el estado de sincronización del nodo.
Método	GET

Respuesta ejemplo	{
	"node_id": "node-01",
	"sync_status": "LOCKED",
	"source": "ptp_grandmaster",
	"last_update": "2025-05-29T15:03:00.000Z",
	"offset_ns": 12,
	"drift_ppm": 0.02
	}

4. `POST /api/time/sync-request`

Descripción	Solicita información de sincronización a otro nodo.
Método	POST

Cuerpo de solicitud	{
	"target_node": "node-02",
	"request_type": "ping_time"
	}

Respuesta ejemplo	{
	"target_node": "node-02",
	"timestamp": "2025-05-29T15:06:10.020Z",
	"offset_ns": 9,
	"signed": true,
	"signature": "MEQCID1Rg9q..."
	}

5.2.2. Infrastructure as a Service (IaaS)

Un posible servicio *Infrastructure as a Service* podría basarse en la creación automatizada de máquinas virtuales dentro de los sistemas de la red, ya preconfiguradas para integrarse con la red de sincronización temporal. Esta automatización puede lograrse mediante el uso de *templates* en el hipervisor Proxmox.

Para ello, se debe crear una máquina virtual base que tenga la capacidad de conectarse correctamente a la red de sincronía. Esta máquina deberá incluir los paquetes necesarios para la sincronización temporal, como `ptp4l` (también conocido como `linuxptp` en sistemas Debian) para el caso de PTP, o `chrony` si se prefiere utilizar NTP.

Además, es imprescindible que el servicio de sincronización esté configurado para ejecutarse automáticamente como demonio, de modo que la máquina comience a sincronizar su reloj en cuanto arranque, sin requerir intervención adicional por parte del usuario.

```
[global]
# Use hardware time stamping
tx_timestamp_type     hardware
rx_timestamp_type     hardware
clockClass            248
clockAccuracy         0xFE
offsetScaledLogVariance 0xFFFF
priority1             128
priority2             128
domainNumber          0
logging_level         6
```

Listing 5.1: Ejemplo del archivo /etc/linuxptp/ptp4l.conf contenido en el template de Proxmox VE

```
[Unit]
Description=PTP IEEE 1588 daemon (ptp4l)
After=network.target

[Service]
ExecStart=/usr/sbin/ptp4l -f /etc/linuxptp/ptp4l.conf -i enp3s0 -m
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Listing 5.2: Ejemplo del archivo /etc/systemd/system/ptp4l.service contenido en el template de Proxmox VE

```
server 0.pool.ntp.org iburst
server 1.pool.ntp.org iburst

driftfile /var/lib/chrony/drift
makestep 1.0 3
rtcsync
logdir /var/log/chrony
```

Listing 5.3: Ejemplo del archivo /etc/chrony/chrony.conf contenido en el template de Proxmox VE

```
[Unit]
Description=Chrony NTP Server
After=network.target

[Service]
ExecStart=/usr/sbin/chronyd -F -1
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Listing 5.4: Ejemplo del archivo /etc/systemd/system/ntp.service contenido en el template de Proxmox VE

Después, dentro del hipervisor se puede crear un *template* de esta imagen y, de esta manera, cada vez que se genere una nueva máquina virtual, esta estará conectada automáticamente a la red de sincronía con la tecnología que se requiera.

5.2.3. Platform as a Service (PaaS)

El hipervisor Proxmox VE se basa en la tecnología de virtualización denominada *Kernel-based Virtual Machine* (**KVM**), que permite la ejecución de máquinas virtuales como procesos del espacio de usuario gestionados directamente por el núcleo del sistema operativo Linux. Esta capacidad se complementa con el uso de **QEMU** (Quick Emulator), un emulador de *hardware* de código abierto que proporciona a cada máquina virtual un conjunto completo de dispositivos virtuales —como interfaces de red, controladoras de almacenamiento y periféricos—, simulando un entorno *hardware* completo. La combinación de KVM y QEMU [21] permite ejecutar sistemas operativos invitados con un alto grado de aislamiento y rendimiento, aprovechando las extensiones de virtualización del procesador físico (por ejemplo, Intel VT-x o AMD-V). No obstante, el acceso de las máquinas virtuales al *hardware* físico del *host* se produce de forma indirecta, a través de interfaces virtualizadas, lo cual garantiza la abstracción del entorno y la seguridad del sistema anfitrión.

Utilizando el módulo del kernel de Linux `ptp_kvm`, se puede lograr que las máquinas virtuales del hipervisor accedan directamente al reloj del *host*. Este módulo se encarga de exponer un reloj PTP virtual a través del subsistema de relojes del propio kernel, proporcionando así un nivel de sincronización más preciso que métodos tradicionales como, por ejemplo, **NTP**.

Cuando el módulo está cargado en la máquina virtual, esta expone el reloj del *host* en la ruta `/dev/ptp0`, permitiendo que usuarios avanzados puedan realizar llamadas al sistema mediante interfaces estándar como `ioctl()` [22] para obtener marcas de tiempo de alta precisión.

5.2.4. Software as a Service (SaaS)

La idea de *Software as a Service* es proporcionar al usuario un programa que tenga la capacidad de devolverle la hora de la red sin necesidad de ninguna preparación previa. Asumiendo que en 5.2.3 se explica cómo integrar un descriptor para que todas las máquinas virtuales, a continuación, en 5.5, se proporcionará un código en Rust que abre el descriptor */dev/ptp0*, lee sobre él y devuelve por salida estándar la hora en formato UTC.

```rust
use std::fs::File;\lstset{
use std::io;
use std::mem;
use std::os::unix::io::AsRawFd;
use std::time::{SystemTime, UNIX_EPOCH};
use chrono::{DateTime, Utc};

const PTP_CLOCK_GETTIME: u32 = 0xC0043B05;

#[repr(C)]
#[derive(Debug)]
struct PtpClockTime {
    sec: i64,
    nsec: u32,
}

fn read_ptp_time() -> io::Result<DateTime<Utc>> {
    let file = File::open("/dev/ptp0")?;
    let fd = file.as_raw_fd();

    let mut ts = PtpClockTime { sec: 0, nsec: 0 };

    let ret = unsafe {
        libc::ioctl(fd, PTP_CLOCK_GETTIME as libc::c_ulong, &mut ts)
    };

    if ret < 0 {
        return Err(io::Error::last_os_error());
    }

    let duration = std::time::Duration::new(ts.sec as u64, ts.nsec);
    Ok(DateTime::<Utc>::from(UNIX_EPOCH + duration))
}

fn main() {
    match read_ptp_time() {
        Ok(utc_time) => {
            println!("{}", utc_time.to_rfc3339_opts(chrono::SecondsFormat::Nanos, true));
        }
        Err(_) => {
            match SystemTime::now().duration_since(UNIX_EPOCH) {
                Ok(duration) => {
                    let datetime = DateTime::<Utc>::from(UNIX_EPOCH + duration);
                    println!("{}", datetime.to_rfc3339_opts(chrono::SecondsFormat::Nanos, true
                        ));
                }
                Err(e) => eprintln!("Error: {}", e),
            }
        }
    }
}
```

Listing 5.5: Programa en Rust que devuelve la hora sincronizada en formato UTC abriendo el descripttor explicado en Platform as a Service

Capítulo 6

Integración de tecnologías de sincronización y evaluación de los resultados

A continuación, se explicará cómo se ha realizado la disposición de pruebas dentro del entorno virtualizado de pruebas. Se adjuntarán capturas de cómo se ha gestionado este entorno y cómo se ha dispuesto para realizar las pruebas.

6.1. Métricas de evaluación

Para poder realizar una comparativa sobre la mejor tecnología disponible en el momento, se ha realizado un experimento utilizando uno de los *racks* del nodo **Cedint**; en él, se han enracado los siguientes dispositivos para hacer comparativas de las tecnologías explicadas en 2.

- **Secure Sync 2400:** Este dispositivo es un equipo de distribución y generación de tiempo preciso basado en una referencia GNSS, que disciplina un oscilador interno. A partir de esta referencia estable, puede generar simultáneamente múltiples salidas de tiempo en distintos formatos, incluyendo señal de sincronía 10 MHzPPS, IRIG-B, NTP y PTP [11].
- **White Rabbit Switch Low Jitter:** Este dispositivo de conmutación diseñado para redes de sincronización temporal de alta precisión. Implementa el protocolo White Rabbit (WR), una extensión de PTP basada en *hardware*, que permite una sincronización subnanosegundo entre nodos conectados.
- **Servidores enracables Supermicro:** Modelo AS-1014S-WTRT-EU, suficiente para soportar las pruebas diseñadas. Estos nodos se han llamado **quasar, quark y qubata** y están interconectados en un mismo clúster de Proxmox.

Dentro de cada uno de los ordenadores Supermicro, se instaló el sistema operativo Proxmox para poder hacer distintas pruebas entre máquinas virtuales. La señal recibe los datos de manera externa y se pretende ver la calidad que pueden otorgar estas máquinas virtuales utilizando la frecuencia y el *offset*. En la fotografía 6.2 se puede observar dentro de cada nodo un cliente PTP y un cliente NTP.

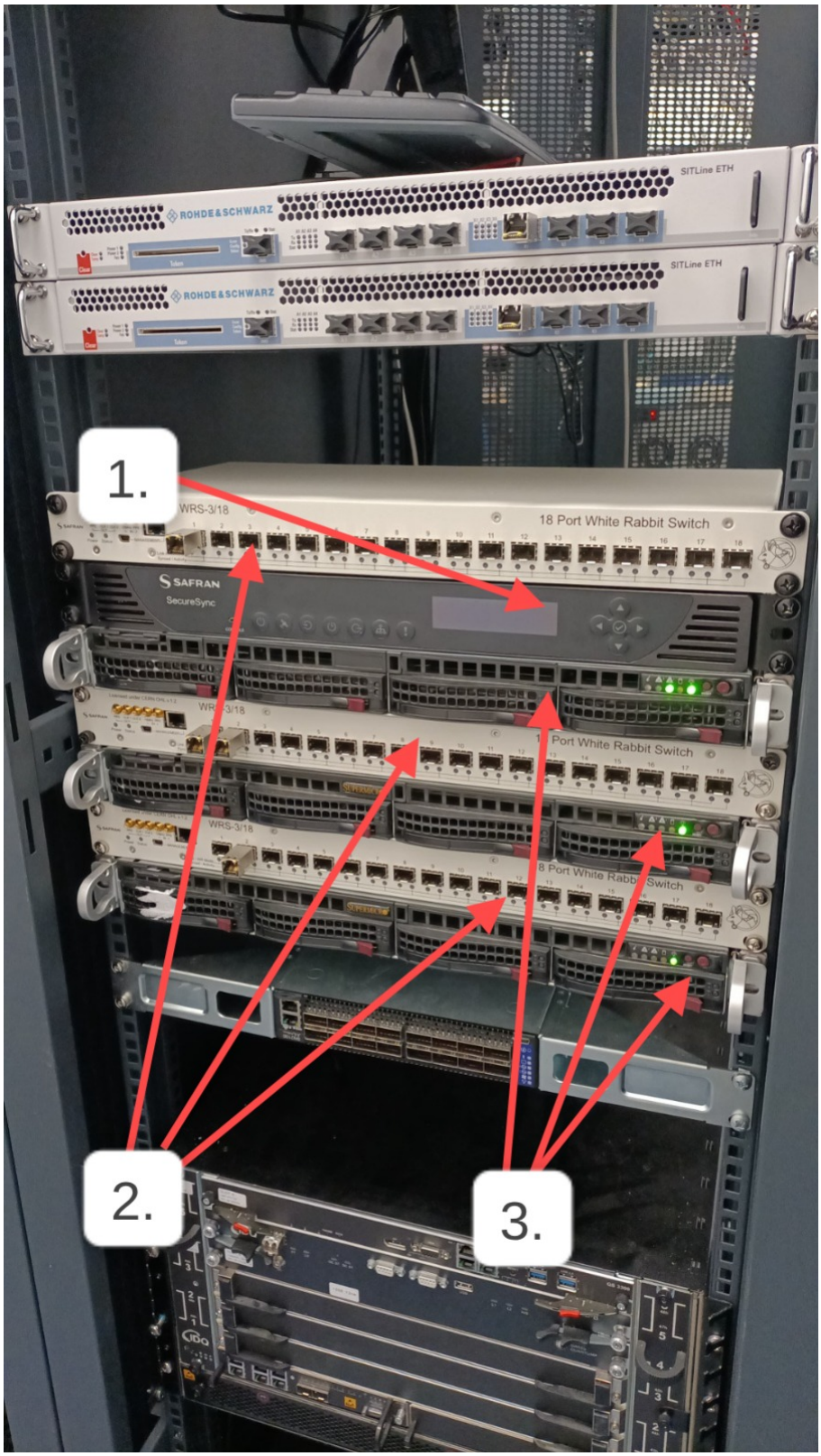

Figura 6.1: Configuración del rack:
1. Fuente de tiempos Secure Sync 2400
2. White Rabbit Switch Low Jitter
3. Servidores enracables Supermicro.

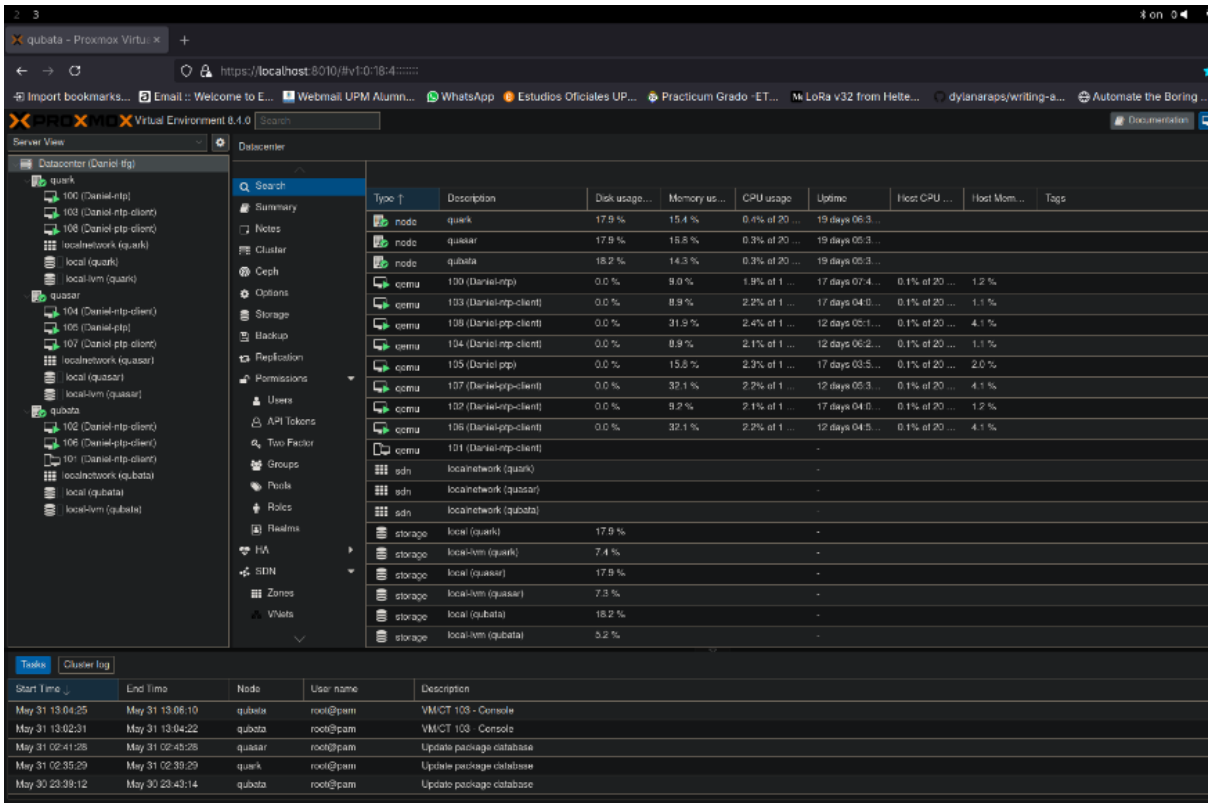

Figura 6.2: Configuración del experimento: quark contiene la máquina virtual que hace de *Master clock* de NTP que utiliza la señal del reloj oficial de Debian; quasar hospeda la máquina que hace de *Master clock* de PTP con la misma señal maestra. Todas tienen un cliente de cada una. Las máquinas cliente han recogido métricas de los datos recibidos para poder ver la calidad de la señal de estos tiempos utilizando la frecuencia y el *offset*.

6.2. Comparativa de NTP

A continuación se presentan las fluctuaciones observadas durante el experimento, analizando primero los resultados obtenidos en las métricas constantes y, posteriormente, los correspondientes a las métricas pseudoaleatorias.

6.2.1. Comparativa de frecuencias por segundo

El análisis de la frecuencia de corrección (*frequency_ppm*) en tres nodos virtualizados bajo Proxmox revela cómo el entorno físico de cada máquina virtual influye significativamente en la estabilidad del reloj y, por tanto, en la eficacia de la sincronización mediante NTP.

El primer nodo (Figura 6.3) está virtualizado en el mismo entorno físico que el reloj maestro. Esta condición elimina prácticamente la latencia de red y mejora la estabilidad térmica y de CPU, lo que permite que el reloj virtualizado requiera muy pocas correcciones. El resultado es una sincronización extremadamente precisa con ajustes mínimos, como muestra la gráfica, con una frecuencia casi constante y sin fluctuaciones destacables.

El segundo nodo (Figura 6.4), alojado en una máquina diferente, presenta una serie de fluctuaciones, escalones y transiciones en la frecuencia. Este comportamiento es típico en entornos donde el reloj virtual está sometido a interferencias, ya sea por migraciones, cambios en la asignación de CPU o por la carga variable del hipervisor. Aunque la sincronización sigue siendo funcional, el demonio NTP debe adaptarse constantemente, lo que se refleja en las correcciones dinámicas observadas.

Por último, el tercer nodo (Figura 6.5), también en un entorno virtual distinto al del maestro, muestra una frecuencia estable en torno a los 5,5 ppm. Aunque esta frecuencia es mayor, su constancia sugiere que el reloj del sistema presenta una deriva regular que NTP corrige de forma eficiente. Este caso representa un escenario intermedio: no tan óptimo como el del nodo colocalizado con el maestro, pero con condiciones lo suficientemente estables como para mantener la sincronización sin grandes sobresaltos.

6.2.2. Comparativa de frecuencias pseudoaleatorias

A diferencia de las gráficas anteriores, en este caso las muestras de `frequency_ppm` han sido tomadas en momentos puntuales y pseudoaleatorios, por lo que el análisis no se centra en la evolución temporal de la sincronización, sino en la comparación entre comportamientos instantáneos de los diferentes nodos virtualizados.

En el caso del nodo 1 (Figura 6.6), la frecuencia se concentra firmemente en torno a los 0,6 ppm, lo cual coincide con observaciones anteriores de nodos que comparten el entorno físico con el reloj maestro. Este valor sugiere una sincronización madura y estable, con un reloj local que apenas requiere ajustes en ese momento.

El nodo 2 (Figura 6.7) muestra una mayor dispersión, con frecuencias que oscilan entre 1,5 y 2,5 ppm. Este rango, aunque moderado, es indicativo de un entorno más dinámico o con menor aislamiento del reloj. Podría reflejar momentos de carga del hipervisor, actividad de la VM o reacciones ante pequeñas desviaciones acumuladas.

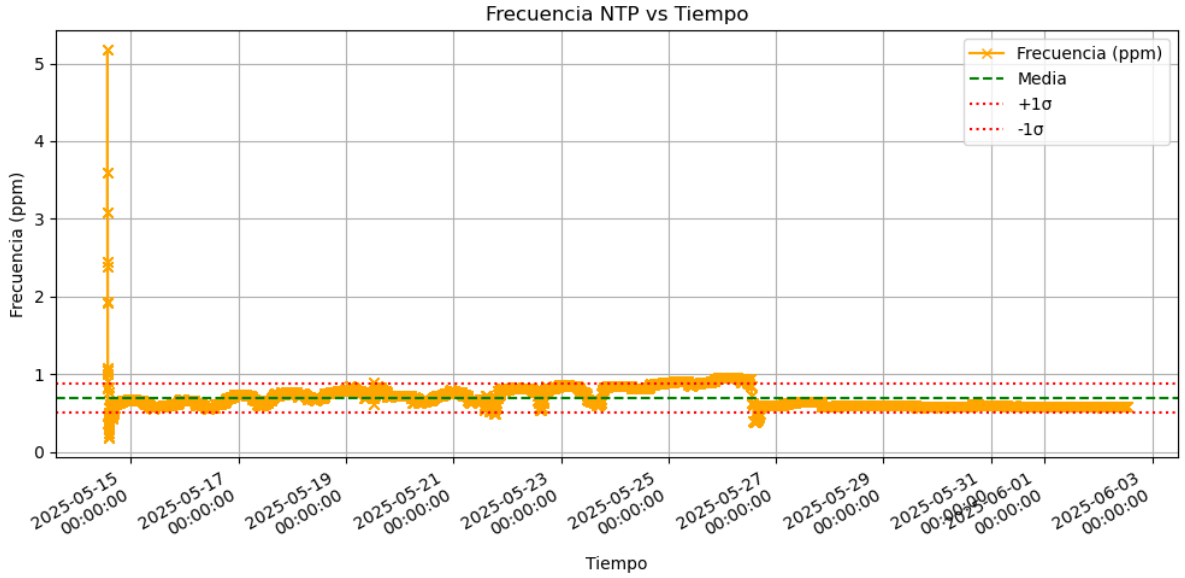

Figura 6.3: Frecuencia Nodo 1: frecuencia estabilizada en torno a 0,6 ppm. Este cliente estaba virtualizado en el mismo nodo que el *Master*.

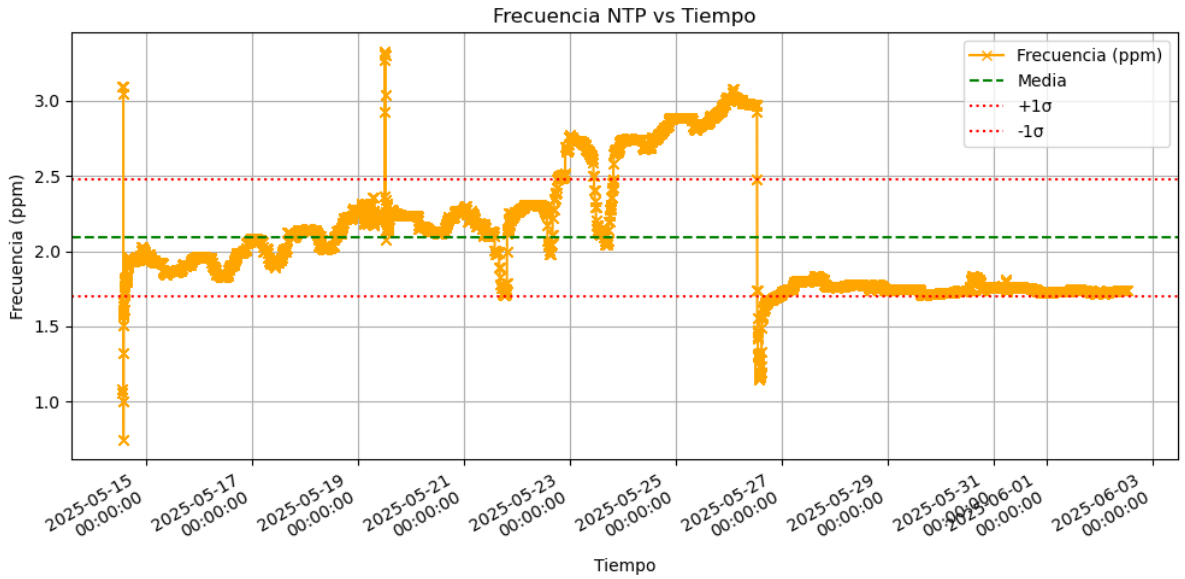

Figura 6.4: Frecuencia Nodo 2: frecuencia variable con saltos progresivos y fases de reajuste visibles.

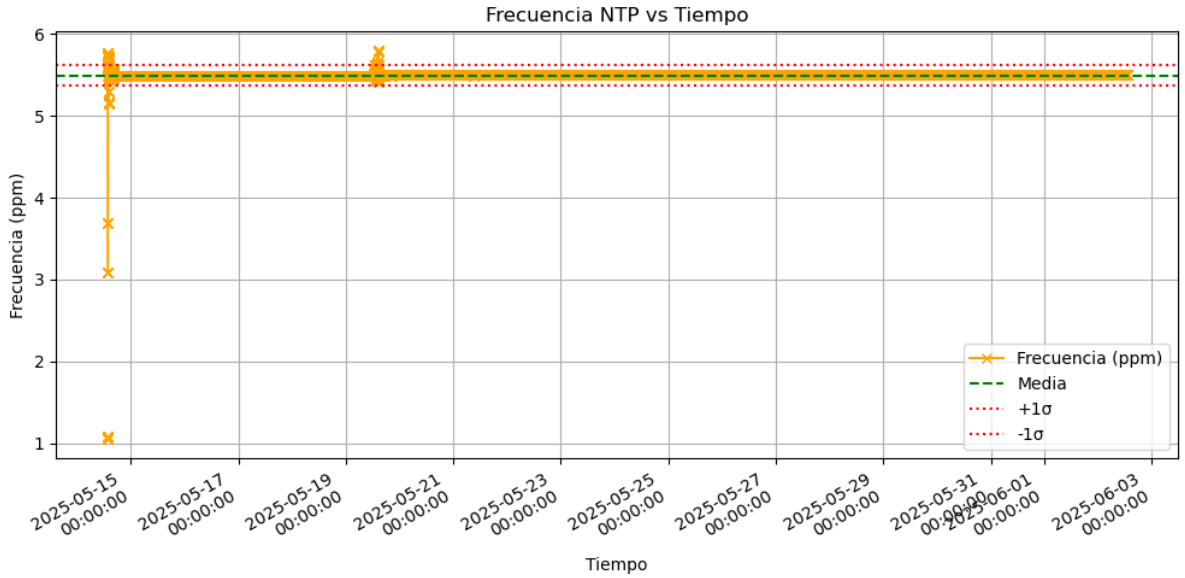

Figura 6.5: Frecuencia Nodo 3: frecuencia constante cercana a 5,5 ppm, con mínimos ajustes a lo largo del tiempo. Cliente estable pero ubicado fuera del *Master*.

El nodo 3 (Figura 6.8) vuelve a mostrar un patrón muy característico: una frecuencia centrada en los 5,5 ppm con muy poca variación. Este valor elevado pero constante es coherente con un reloj local que tiende a desviarse a un ritmo estable y conocido, lo que permite a NTP mantenerlo bajo control sin necesidad de cambios bruscos.

Aunque las muestras sean aleatorias, se observan patrones coherentes con los obtenidos en observaciones continuas: el nodo cercano al maestro sigue mostrando los mejores valores de estabilidad, mientras que los nodos externos presentan características propias de entornos virtuales no deterministas. Estos datos refuerzan la hipótesis de que la estabilidad del entorno físico donde se ejecuta la máquina virtual tiene un impacto directo sobre la calidad de la sincronización NTP, incluso en observaciones aisladas.

6.2.3. Comparativa de *offset* en cada segundo

Las gráficas 6.9, 6.10 y 6.11 representan la evolución del `offset` (diferencia temporal entre el reloj local y el reloj maestro) medida por el cliente NTP en tres máquinas virtuales distintas, alojadas en un entorno Proxmox.

En las tres gráficas se observa un patrón común: la mayor parte del tiempo el *offset* permanece por debajo de $1{\cdot}10^{-3}$ segundos (1 milisegundo), con excepción de algunos picos al inicio del periodo o en momentos aislados. Este comportamiento es un indicador claro de que el sistema de sincronización está funcionando con precisión razonable, lo cual es destacable tratándose de máquinas virtuales.

El buen funcionamiento de estos *offsets* se debe a la configuración de Proxmox, que permite, al estar las **máquinas virtuales** en un mismo *cluster*, reducir la latencia y así conseguir estos resultados.

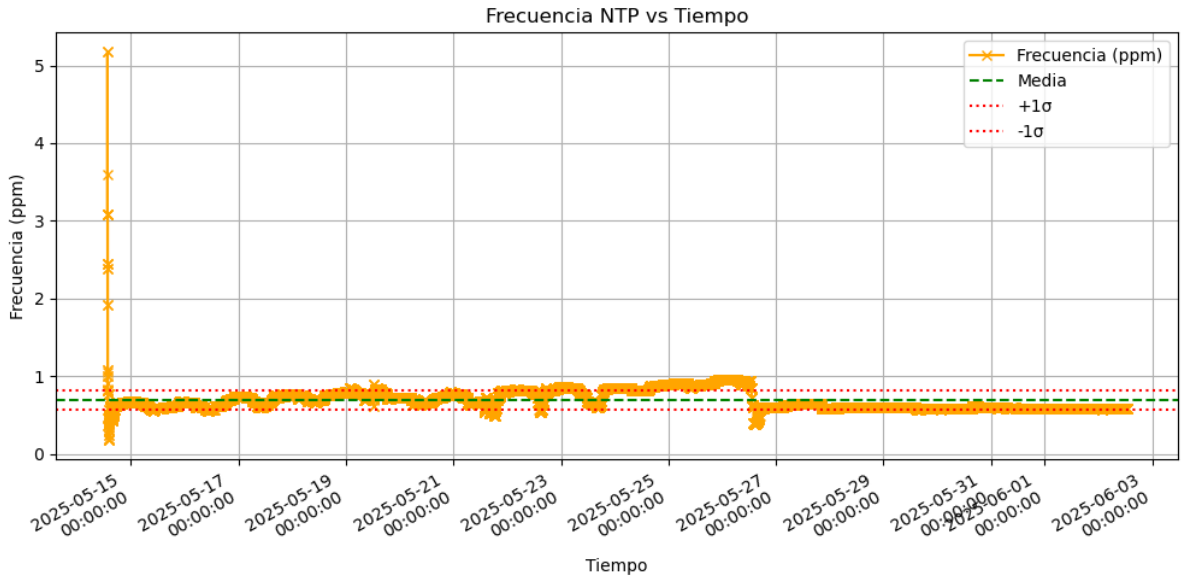

Figura 6.6: Frecuencia Nodo 1: frecuencia estabilizada en torno a 0,6 ppm. Este cliente estaba virtualizado en el mismo nodo físico que el *Master*, lo que explica la baja variabilidad y la alta estabilidad.

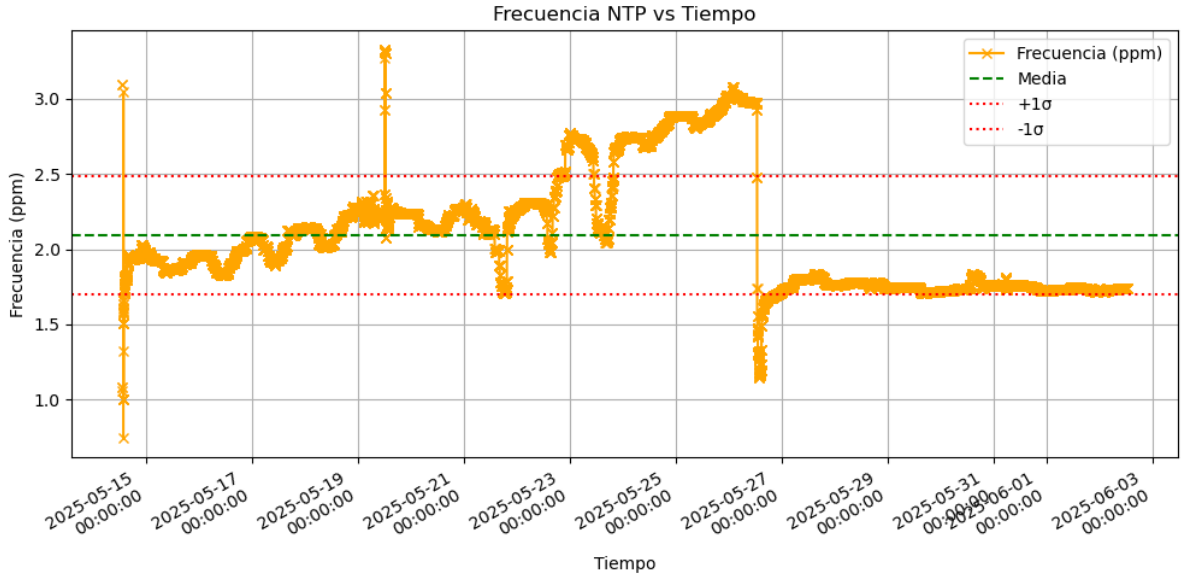

Figura 6.7: Frecuencia Nodo 2: frecuencia más dispersa, oscilando entre 1,5 y 2,5 ppm. El comportamiento refleja cierta inestabilidad, posiblemente inducida por el entorno virtual en un nodo distinto al del *Master*.

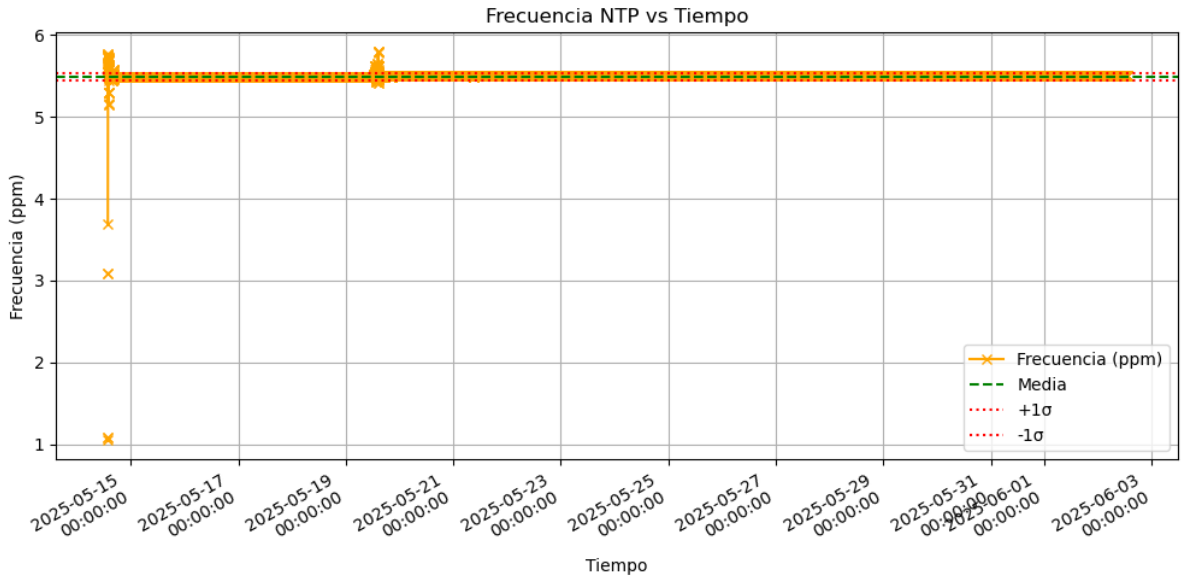

Figura 6.8: Frecuencia Nodo 3: frecuencia estable centrada en 5,5 ppm. Aunque fuera del entorno físico del *Master*, este cliente mantiene una deriva regular y controlada del reloj.

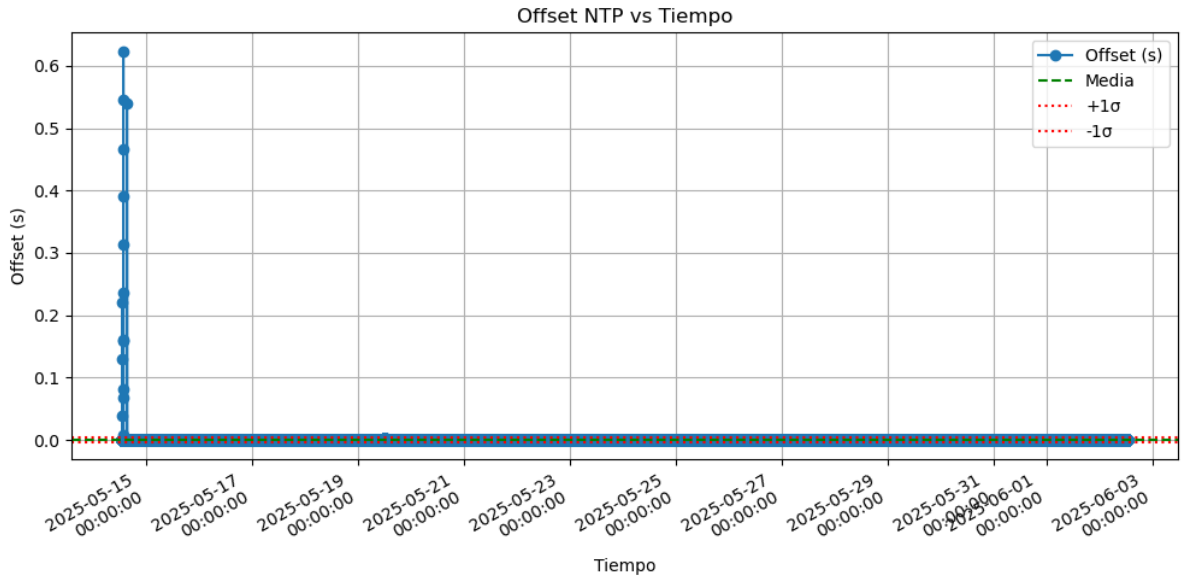

Figura 6.9: Offset Nodo 1: offset estabilizado por debajo de $1{\cdot}10^{-3}$ s tras un arranque inicial con picos de hasta $6{\cdot}10^{-1}$ s. Cliente virtualizado en el mismo host que el *Master*.

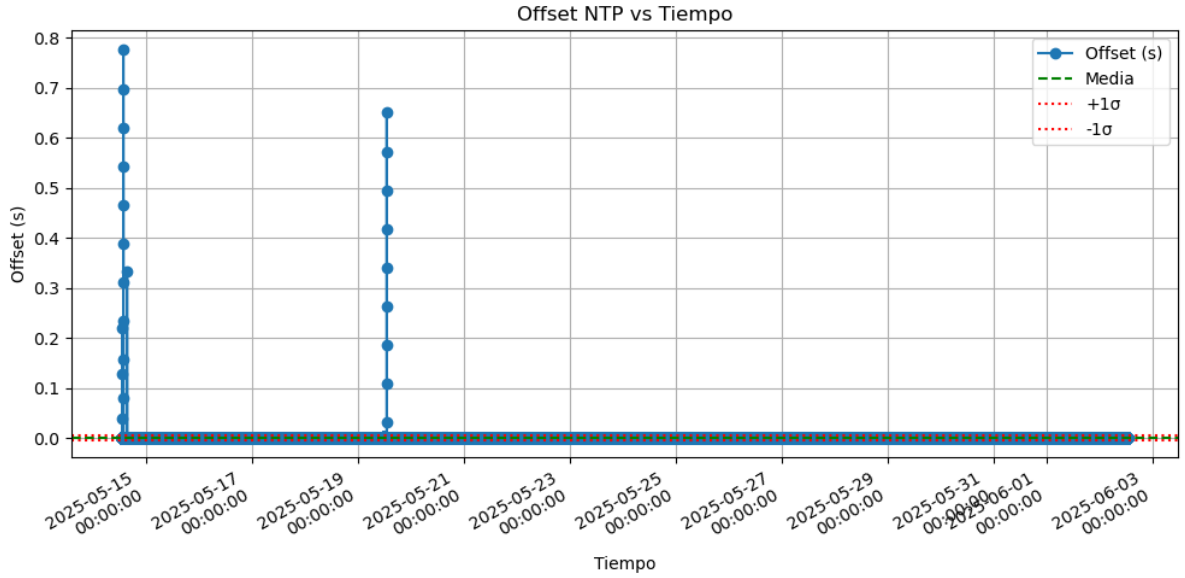

Figura 6.10: Offset Nodo 2: eventos puntuales de hasta $7{\cdot}10^{-1}$ s y otro pico aislado de $6.5{\cdot}10^{-1}$ s, seguidos de un periodo estable. Cliente en entorno distinto al del *Master*.

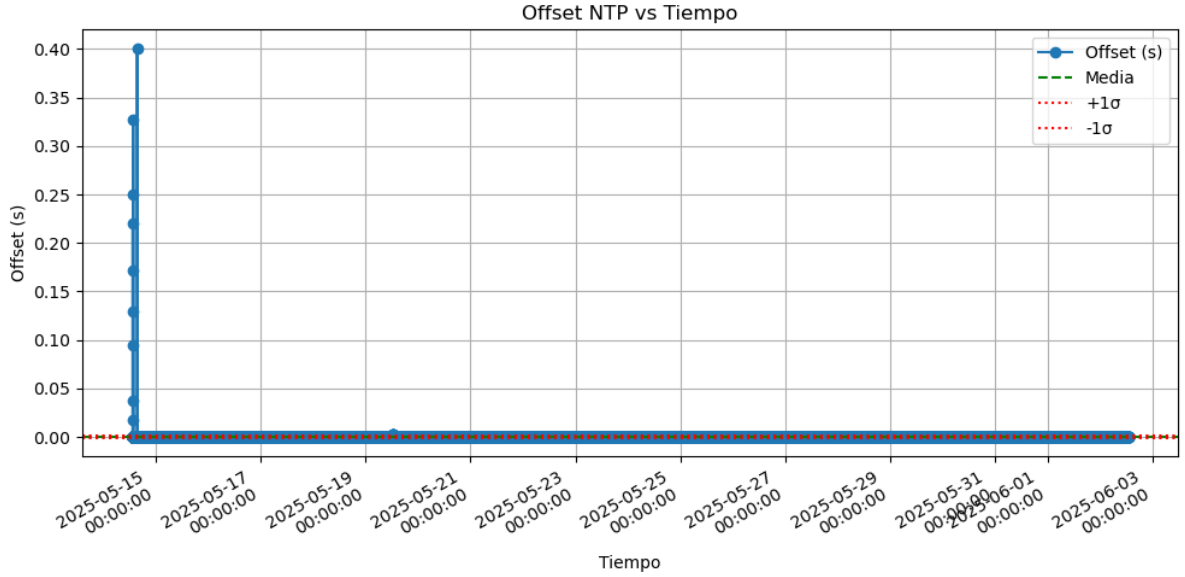

Figura 6.11: Offset Nodo 3: pico inicial de hasta $4{\cdot}10^{-1}$ s y luego estabilidad total por debajo de $1{\cdot}10^{-3}$ s. Comportamiento consistente con una deriva bien corregida.

6.2.4. Comparativa de *offset* pseudoaleatoria

Las siguientes gráficas representan muestras del `offset`, recogidas en momentos aleatorios a lo largo de varios días en tres máquinas virtuales distintas bajo Proxmox.

Aunque no se dispone de una serie continua, los resultados ofrecen una visión general del rendimiento de la sincronización en distintos momentos del tiempo. En todos los casos, los valores de *offset* se mantienen por debajo de $2{\cdot}10^{-3}$ s (2 milisegundos), lo cual sigue siendo un resultado muy bueno incluso para entornos físicos, y notable en máquinas virtuales.

El primer nodo (6.12), ubicado en el mismo entorno físico que el maestro, muestra una oscilación general bien contenida, con un pico puntual que alcanza los $1.8{\cdot}10^{-3}$ s. El segundo nodo (6.13) presenta una distribución ligeramente más dispersa, con múltiples valores por encima de $5{\cdot}10^{-4}$ s y picos aislados cercanos a los $1.8{\cdot}10^{-3}$ s. El tercer nodo (6.14) ofrece el comportamiento más estable, con un único pico notable de $1.8{\cdot}10^{-3}$ s y valores residuales casi nulos el resto del tiempo.

Estos valores tan bajos refuerzan la idea de que, si bien la virtualización introduce cierta incertidumbre, una infraestructura bien configurada y una red de baja latencia pueden permitir que NTP mantenga una sincronización precisa y estable incluso en contextos virtuales.

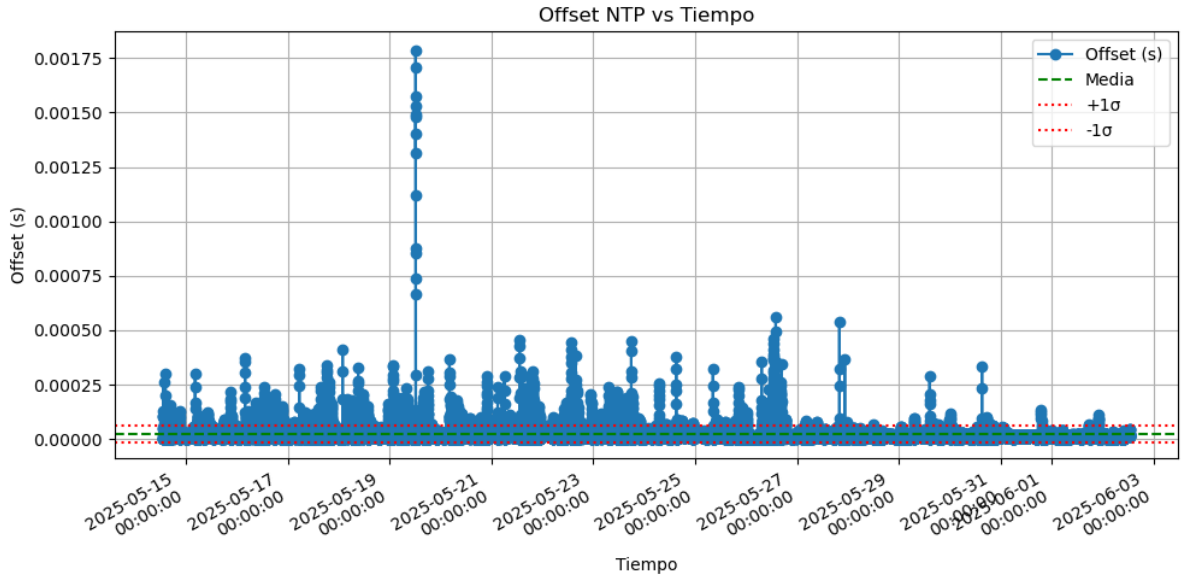

Figura 6.12: Offset Nodo 1: dispersión moderada con un pico puntual de hasta $1.8{\cdot}10^{-3}$ s. Cliente ubicado en el mismo nodo físico que el *Master*.

6.3. Comparativa de PTP

A continuación se observarán las fluctuaciones dentro del experimento realizado, analizando primero los resultados obtenidos en las métricas constantes y, después, los correspondientes a las métricas pseudoaleatorias. Además, se añadirá el *delay* del *path*, con los ajustes de tiempo que eso implica.

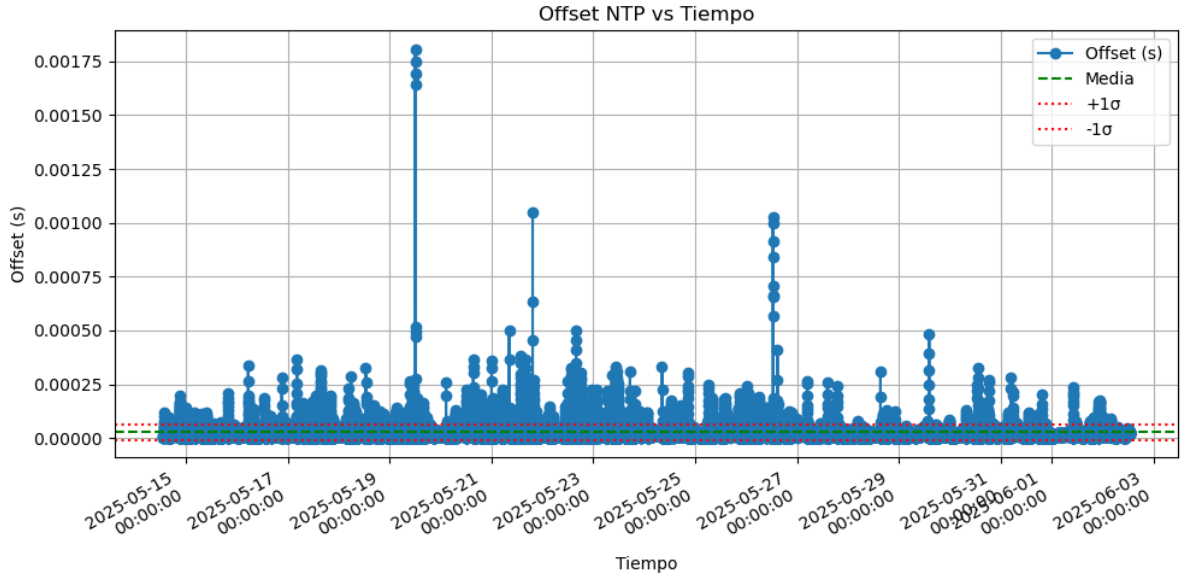

Figura 6.13: Offset Nodo 2: variaciones frecuentes entre $1{\cdot}10^{-4}$ y $5{\cdot}10^{-4}$ s, con picos que alcanzan los $1.8{\cdot}10^{-3}$ s.

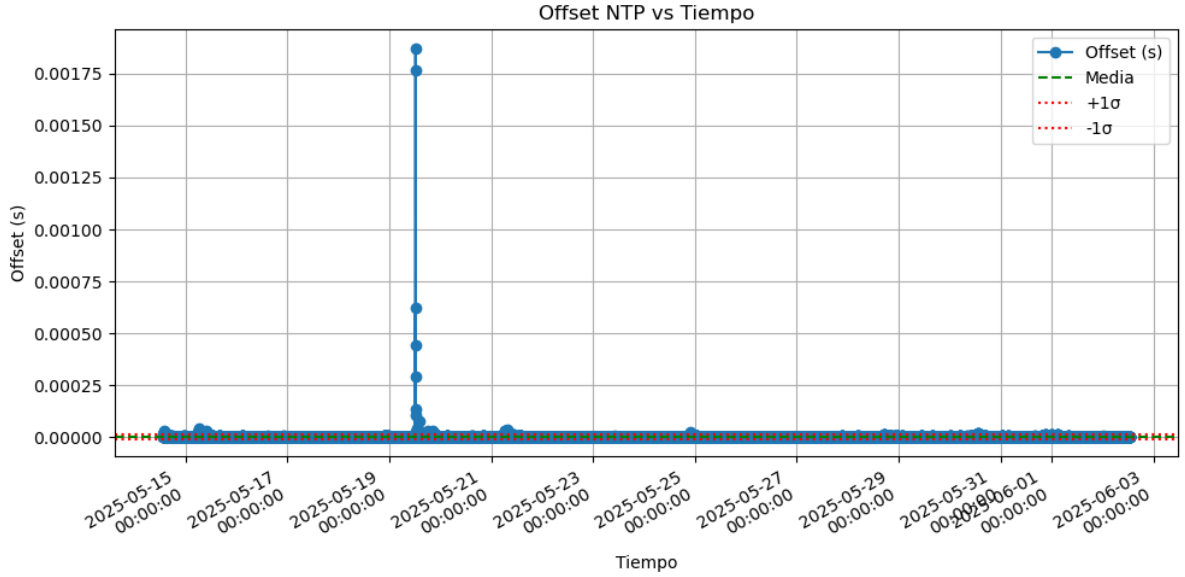

Figura 6.14: Offset Nodo 3: comportamiento mayoritariamente plano con un único pico de $1.8{\cdot}10^{-3}$ s. Cliente alojado en nodo distinto al del maestro.

6.3.1. Comparativa de frecuencias en cada segundo

El análisis de la frecuencia de corrección en los tres nodos clientes del sistema PTP virtualizado bajo Proxmox revela diferencias notables en el comportamiento de sincronización en función del entorno de ejecución.

El primer nodo (6.15) presenta una frecuencia muy variable, con picos extremos en los instantes iniciales y una dispersión considerable a lo largo del experimento. Este patrón sugiere una fase de enganche prolongada y una posterior estabilización menos eficiente, posiblemente relacionada con condiciones menos favorables dentro de su entorno virtual.

El segundo nodo (6.16), que se ejecuta en el mismo entorno físico que el reloj maestro, muestra una frecuencia prácticamente constante durante todo el periodo de captura. La ausencia de oscilaciones notables indica una sincronización rápida y precisa, sin necesidad de correcciones significativas una vez establecido el enlace con el reloj de referencia.

El tercer nodo (6.17), también virtualizado en un entorno distinto al maestro, evidencia un comportamiento intermedio. Tras algunos picos iniciales de corrección, la frecuencia se estabiliza en un rango estrecho y sin variaciones bruscas. Esta evolución sugiere un acoplamiento más favorable que el del primer nodo, aunque sin alcanzar la precisión observada en el segundo.

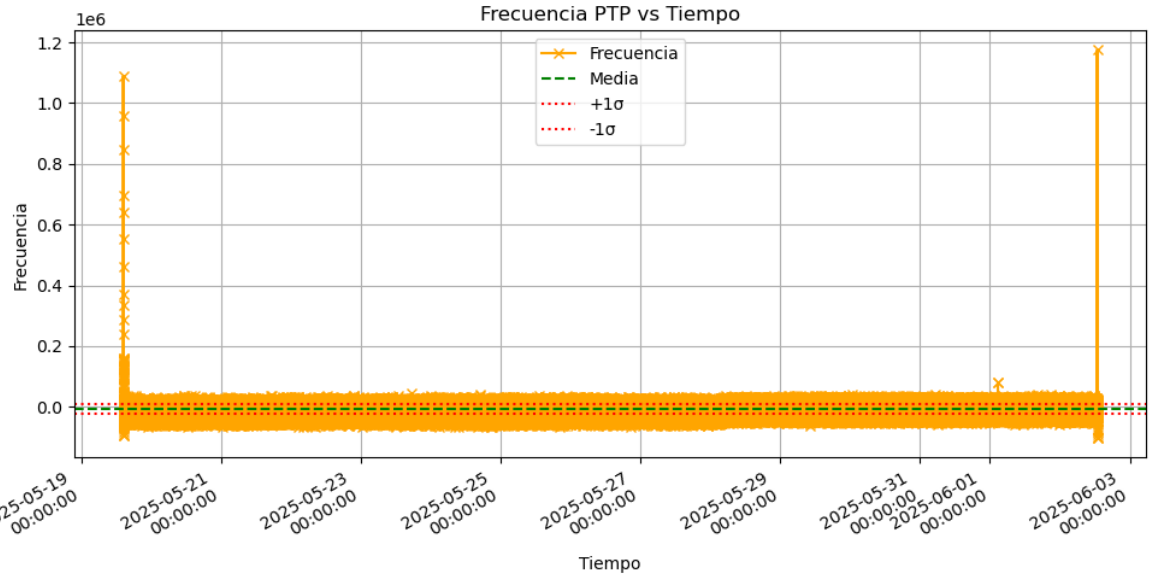

Figura 6.15: Frecuencia Nodo 1: fuerte dispersión inicial y picos extremos, seguidos de una estabilización progresiva. Nodo con mayores dificultades para acoplarse al maestro.

6.3.2. Comparativa de frecuencias pseudoaleatorias

Las muestras aleatorias de la frecuencia PTP reflejan un comportamiento variable según el nodo y el instante de captura. A diferencia de las series continuas, estas

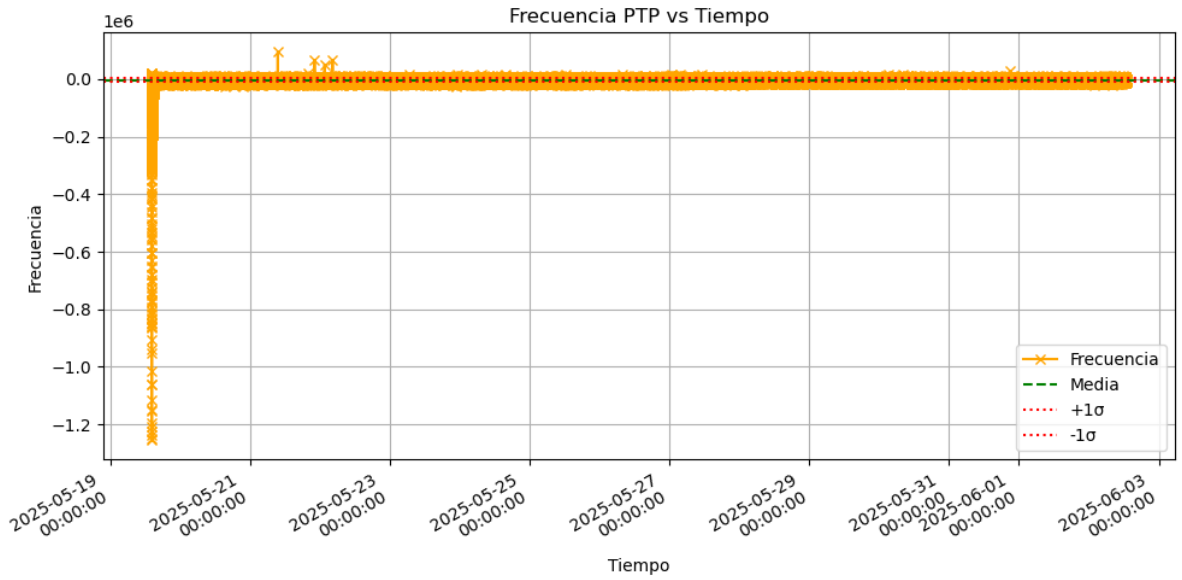

Figura 6.16: Frecuencia Nodo 2: frecuencia casi constante durante toda la captura. Cliente ejecutándose en el mismo entorno físico que el maestro, con sincronización inmediata y estable.

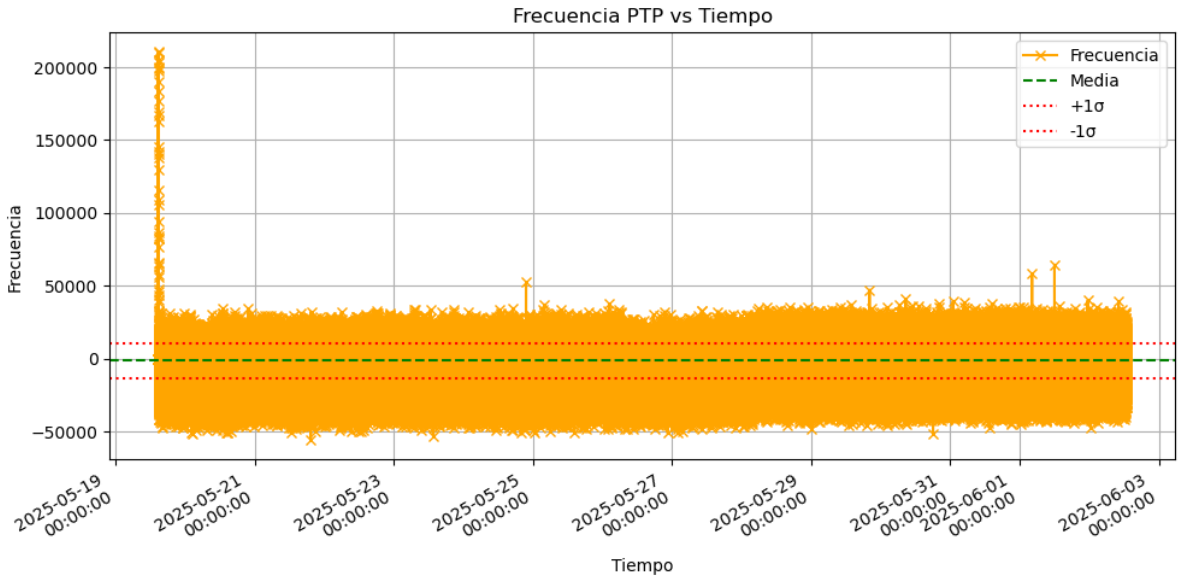

Figura 6.17: Frecuencia Nodo 3: ligeros picos iniciales seguidos de una estabilización rápida. Nodo bien sincronizado, aunque virtualizado en entorno distinto al maestro.

mediciones no permiten identificar tendencias temporales prolongadas, pero ofrecen una visión representativa del rendimiento del sistema en momentos puntuales.

El primer nodo (6.18), virtualizado en el mismo entorno físico que el maestro, presenta frecuencias muy cercanas a cero, con una dispersión mínima. Este resultado indica una sincronización muy precisa, incluso cuando las muestras no siguen un patrón continuo, y confirma que la colocalización con el maestro favorece la estabilidad.

En el segundo nodo (6.19), alojado en una máquina diferente, se observan picos de corrección extremos en los instantes iniciales, algunos superiores a ±1.000.000. Estos valores atípicos sugieren que el cliente aún no estaba completamente sincronizado en el momento de la muestra. Posteriormente, las frecuencias tienden a estabilizarse, reflejando la acción correctiva del protocolo PTP.

El tercer nodo (6.20) muestra un comportamiento intermedio. Aunque las frecuencias presentan cierta dispersión, la mayoría de los valores se concentran en torno a ±50.000, un margen razonable en un entorno virtualizado. La estabilidad es mayor que en el segundo nodo, pero no alcanza la precisión observada en el primero.

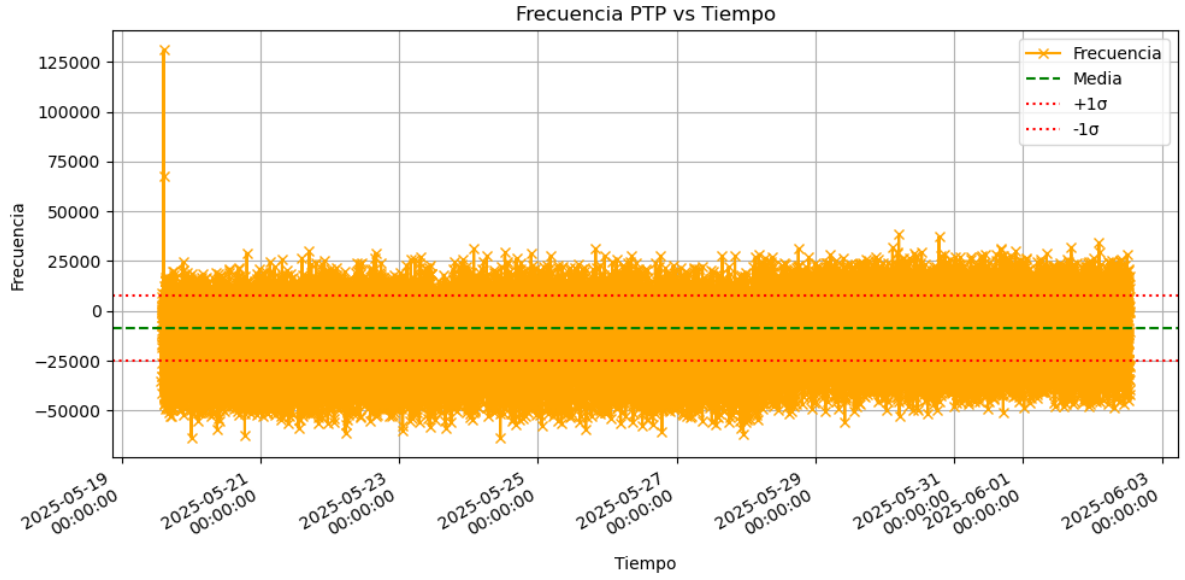

Figura 6.18: Frecuencia Nodo 1: frecuencia centrada en torno a cero con mínima dispersión. Cliente colocalizado con el maestro, lo que favorece la estabilidad incluso en muestras puntuales.

6.3.3. Comparativa de *offset* en cada segundo

Las siguientes gráficas muestran la evolución del offset_ns registrado a lo largo de varios días en tres máquinas virtualizadas bajo Proxmox. Los valores obtenidos se deben al módulo módulo del kernel de Linux `ptp_kvm` explicado en 5.2.3 que al contener todas las máquinas virtuales en el mismo clúster, permite un nivel de sincronía idóneo con el máster.

En general, todos los nodos presentan valores de offset dentro del rango submicrose-

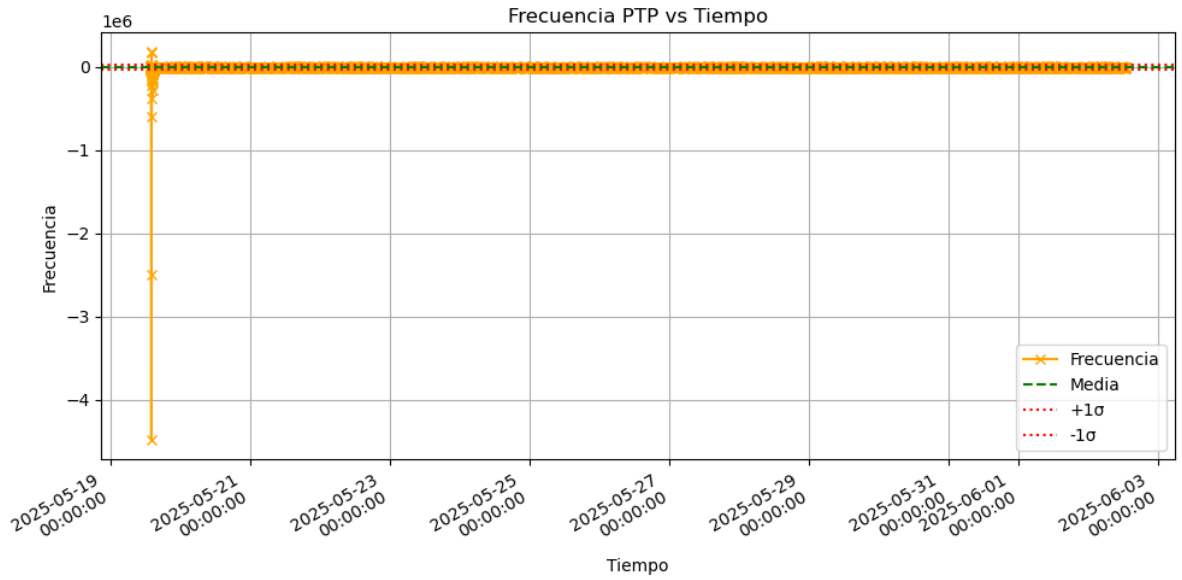

Figura 6.19: Frecuencia Nodo 2: picos extremos de corrección al inicio del registro, típicos de un cliente aún no sincronizado. Posteriormente, la frecuencia converge a valores más estables.

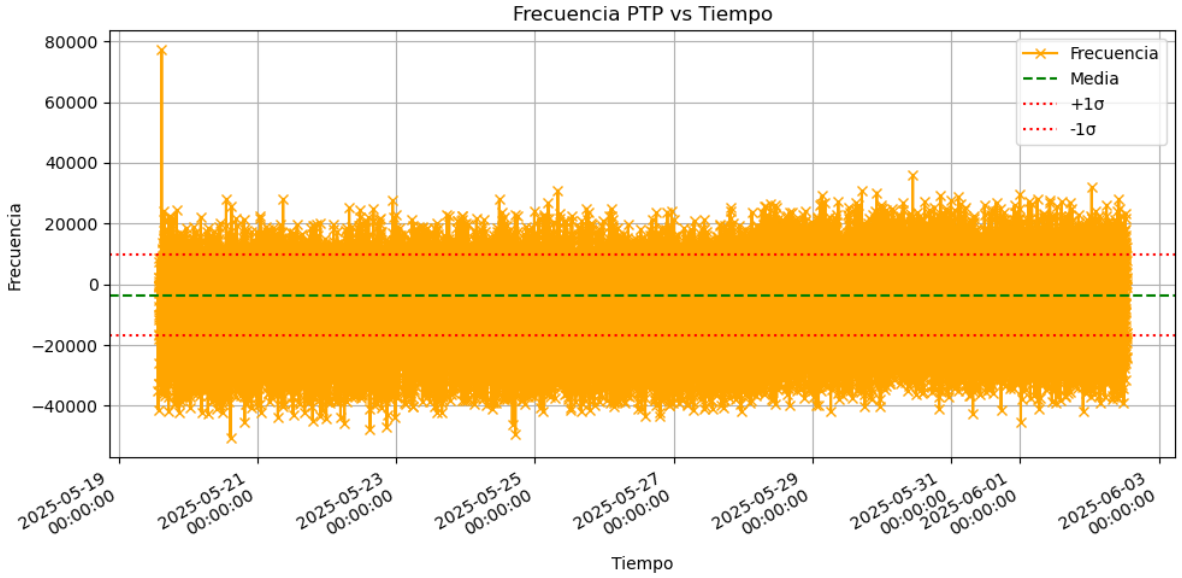

Figura 6.20: Frecuencia Nodo 3: comportamiento intermedio con dispersión moderada. Aunque no tan preciso como el nodo colocalizado, mantiene frecuencias dentro de márgenes aceptables en entorno virtualizado.

gundo, lo que supone una mejora notable respecto a soluciones como NTP, especialmente tratándose de entornos virtuales. El primer nodo (6.21), alojado en un entorno físico distinto al del maestro, mantiene el *offset* controlado en torno a ±150 ns, con una única oscilación puntual al inicio, cercana a los –700 ns. Esta desviación se estabiliza rápidamente y el sistema entra en régimen sin requerir ajustes significativos.

El segundo nodo (6.22), que comparte entorno físico con el reloj maestro, muestra el comportamiento más preciso de los tres. Los valores se distribuyen simétricamente alrededor de cero, con una dispersión mínima y un único pico positivo que apenas supera los 300 ns. Este patrón evidencia una sincronización extremadamente estable, como cabría esperar al eliminar la latencia de red.

Por último, el tercer nodo (6.23), también ubicado en un entorno diferente al maestro, refleja una dinámica similar a la del primero. Tras un pico inicial cercano a los –500 ns, el *offset* se mantiene dentro de un rango estrecho, con un leve sesgo negativo y una dispersión controlada.

Estos resultados confirman que, incluso en entornos virtuales, una infraestructura bien dimensionada y un protocolo como PTP permiten mantener sincronizaciones altamente precisas, con desviaciones temporales muy por debajo de las que se obtendrían con NTP.

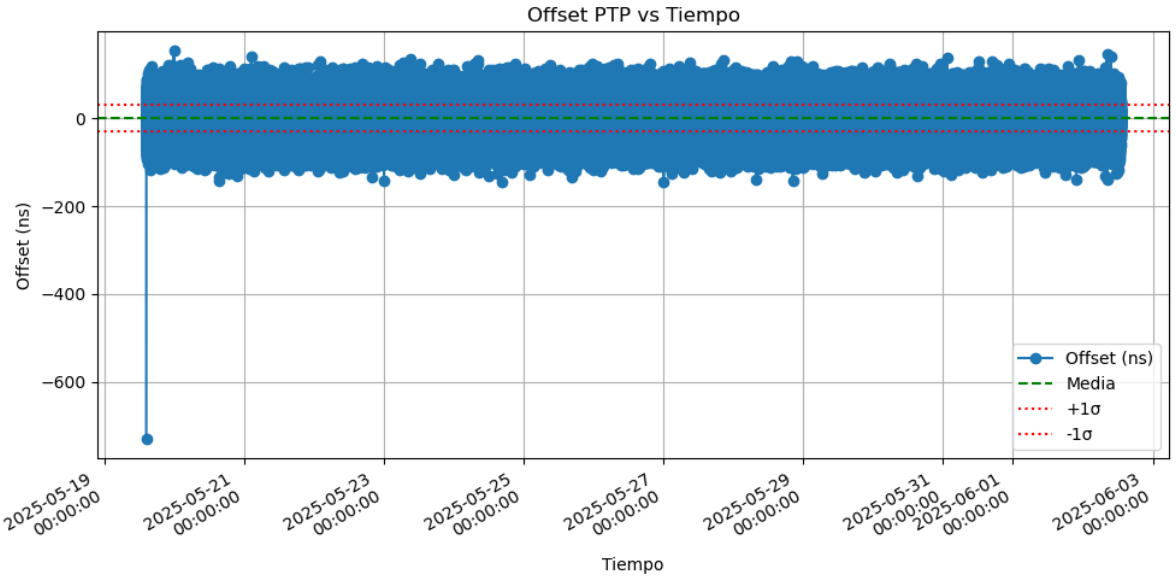

Figura 6.21: Offset Nodo 1: *offset* centrado alrededor de 0 ns, con una oscilación inicial destacada cercana a -700 ns. Tras esta fase, el sistema se estabiliza rápidamente y mantiene una deriva controlada.

6.3.4. Comparativa de *offset* pseudo-aleatoria

En las métricas aleatorias recogidas mediante PTP, se observa que la mayoría de las muestras se encuentran en estado `s0`, lo que indica que el cliente aún no ha alcanzado la fase de sincronización completa con el reloj maestro. Como consecuencia, no se dispone de un valor válido para el *offset*, ya que este solo puede calcularse con

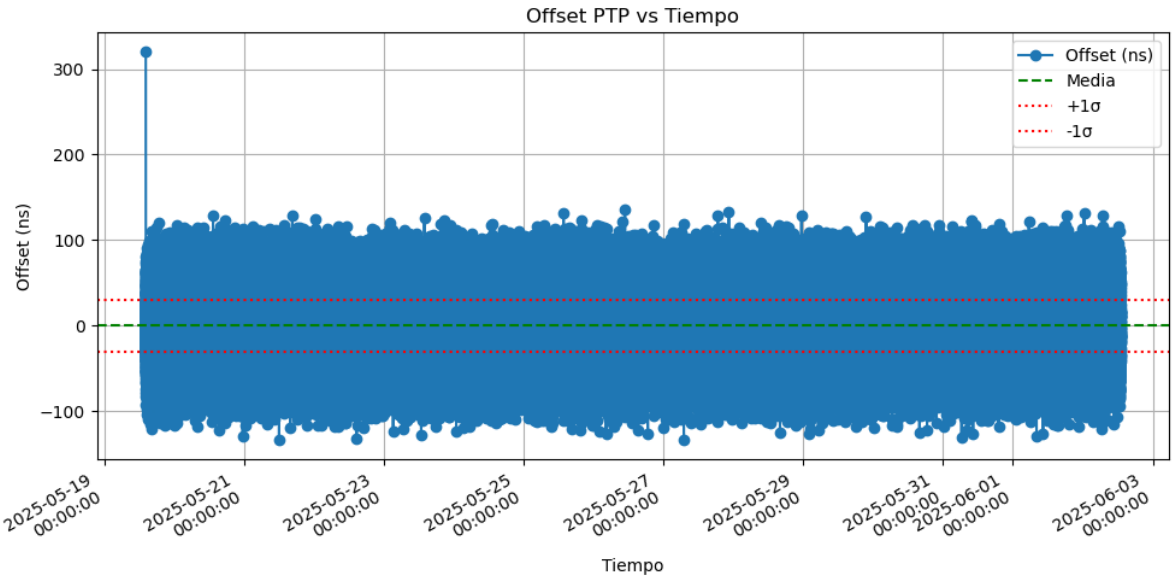

Figura 6.22: Offset Nodo 2: *offset* simétrico y con baja dispersión, con una única oscilación puntual que supera los 300 ns. Nodo virtualizado en el mismo entorno físico que el maestro, con sincronización muy estable.

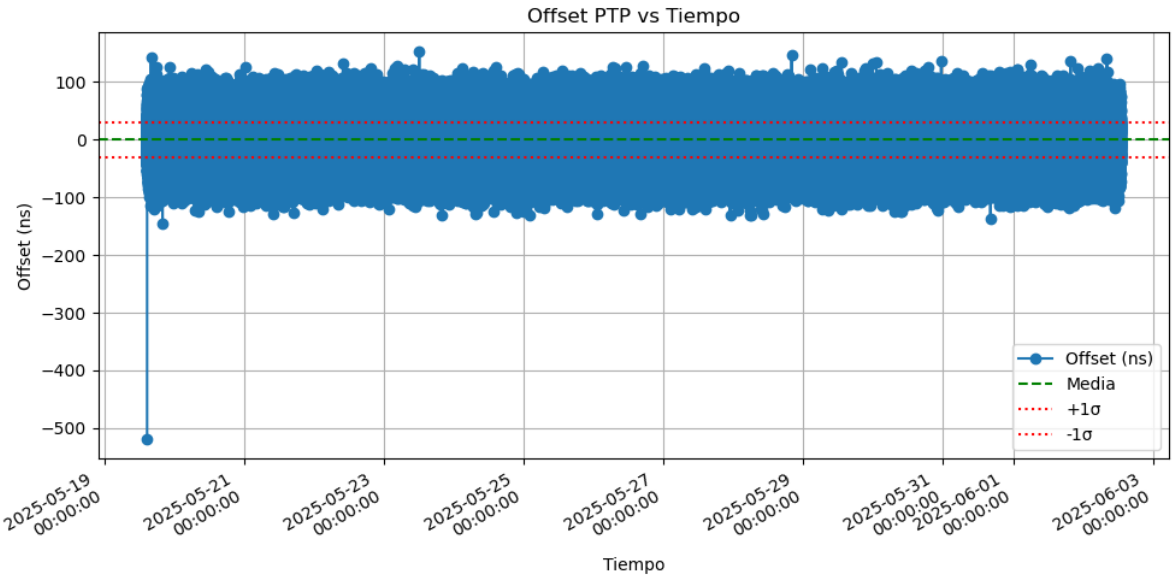

Figura 6.23: Offset Nodo 3: pico inicial negativo cercano a -500 ns seguido de una estabilización sostenida. Aunque con algo más de ruido que el nodo colocalizado con el maestro, el *offset* se mantiene en el rango submicrosegundo.

precisión una vez que el cliente entra en estado `s2` (sincronizado). Esta limitación es inherente al funcionamiento del protocolo PTP, que no proporciona medidas de *offset* fiables mientras no se establece una sincronización estable.

6.3.5. Comparativa de *path delay* en cada segundo

Las gráficas de `path_delay_ns` obtenidas para los tres nodos virtualizados bajo Proxmox reflejan el comportamiento de la latencia de red medida por PTP durante la sincronización. Aunque este parámetro no incide directamente sobre la corrección de frecuencia u *offset*, sí ofrece información clave sobre las condiciones del canal de comunicación entre el reloj maestro y cada cliente.

En el primer nodo, los valores de *path delay* oscilan típicamente entre 400.000 y 700.000 ns, con una transición clara a partir del 29 de mayo, momento en el que la latencia se reduce de forma abrupta. Este salto puede deberse a un cambio en la topología de red virtual, migración del contenedor o reconfiguración de los recursos del hipervisor. En cualquier caso, el sistema continúa operando con estabilidad tras el cambio, lo cual indica que PTP adapta correctamente sus estimaciones.

El segundo nodo muestra valores mucho más bajos, con una media estable en torno a los 190.000 ns. La gráfica es menos dispersa y la desviación estándar más contenida, lo que sugiere una conexión más directa o menos interferida con el reloj maestro. También se observa un descenso de latencia a finales de mayo, aunque menos abrupto que en el primer nodo. Este comportamiento más estable es coherente con la ubicación del nodo en el mismo entorno físico que el maestro.

El tercer nodo presenta una gráfica prácticamente idéntica a la del primero, tanto en rango como en estructura. La coincidencia en los valores y la evolución temporal sugiere que ambos clientes están sometidos a condiciones de red similares, probablemente debido a estar ubicados en *hosts* virtuales distintos al maestro pero dentro del mismo entorno general de red.

En conjunto, las gráficas muestran que, si bien la latencia puede variar de forma significativa en entornos virtualizados, PTP es capaz de seguir ofreciendo una sincronización precisa gracias a su modelo de compensación activa del *delay*.

6.3.6. Comparativa de *path delay* pseudo-aleatoria

Las gráficas de `path_delay_ns` recogidas de forma aleatoria muestran diferencias significativas entre los nodos virtualizados, lo que evidencia cómo la topología de red y las condiciones del entorno de virtualización afectan a la calidad de la sincronización mediante PTP. En el nodo 2 6.28, ubicado en el mismo entorno físico que el reloj maestro, los valores de *delay* se mantienen dentro de un margen estrecho y estable, con una concentración clara en torno a la media y sin desviaciones pronunciadas, reflejando una latencia de red baja y altamente consistente. En cambio, el nodo 1 6.27, virtualizado en un host distinto, presenta un comportamiento anómalo al inicio de la muestra, con un pico de *delay* que supera los 3.5 millones de nanosegundos, seguido de una estabilización rápida. Este tipo de valores extremos podría deberse a condiciones transitorias de la red virtual, como congestión, migraciones de CPU o procesos de arranque. Por su parte, el nodo 3 6.29 muestra una distribución similar a la del nodo 2 en cuanto a la forma general, aunque con una dispersión algo mayor y algunos valores atípicos por debajo del umbral esperado, lo que podría atribuirse a *jitter*

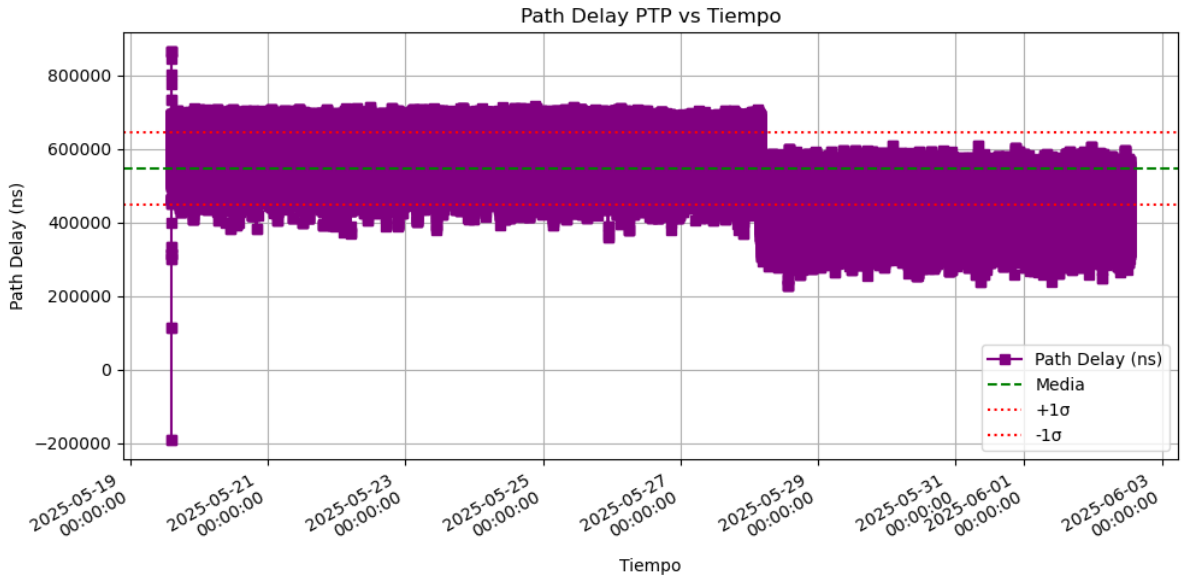

Figura 6.24: Path Delay Nodo 1: Valores entre 400–700 µs, con un cambio brusco de latencia a partir del 29 de mayo. Este salto sugiere una posible reconfiguración o migración virtual.

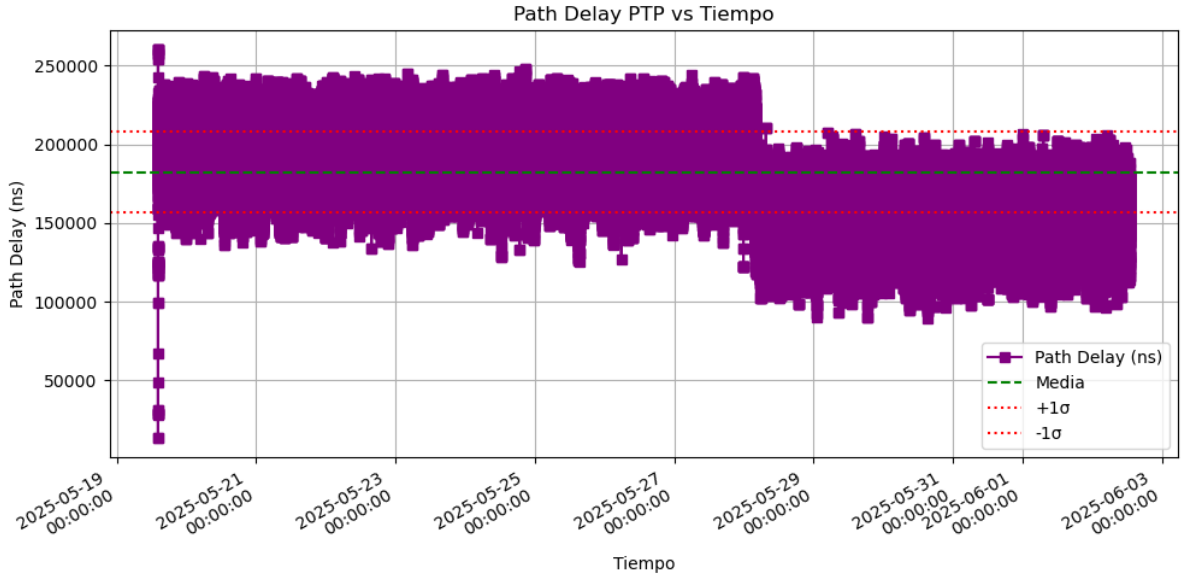

Figura 6.25: Path Delay Nodo 2: Delay promedio estable en torno a 190 µs, con baja dispersión. Nodo colocalizado con el maestro, lo que explica la menor latencia y variabilidad.

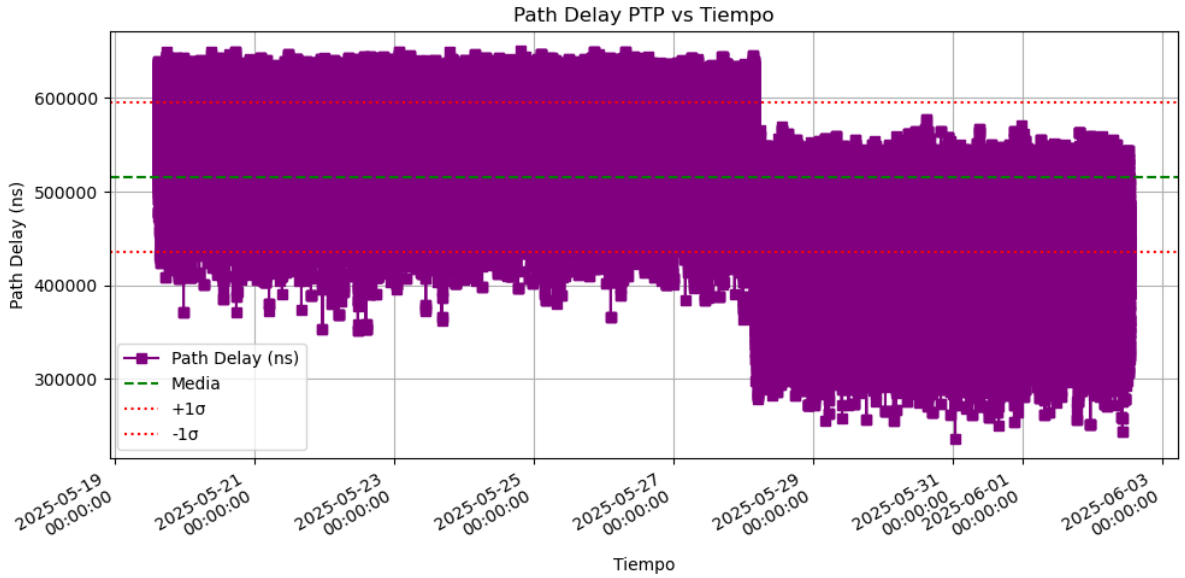

Figura 6.26: Path Delay Nodo 3: Evolución y rango de valores prácticamente idénticos a los del Nodo 1, lo que sugiere condiciones de red similares en entornos virtualizados distintos al maestro.

o variaciones en la calidad del enlace virtualizado. Estos resultados refuerzan el valor del `path_delay` como métrica sensible al rendimiento del canal de sincronización en entornos virtuales, incluso en escenarios con muestreo no continuo.

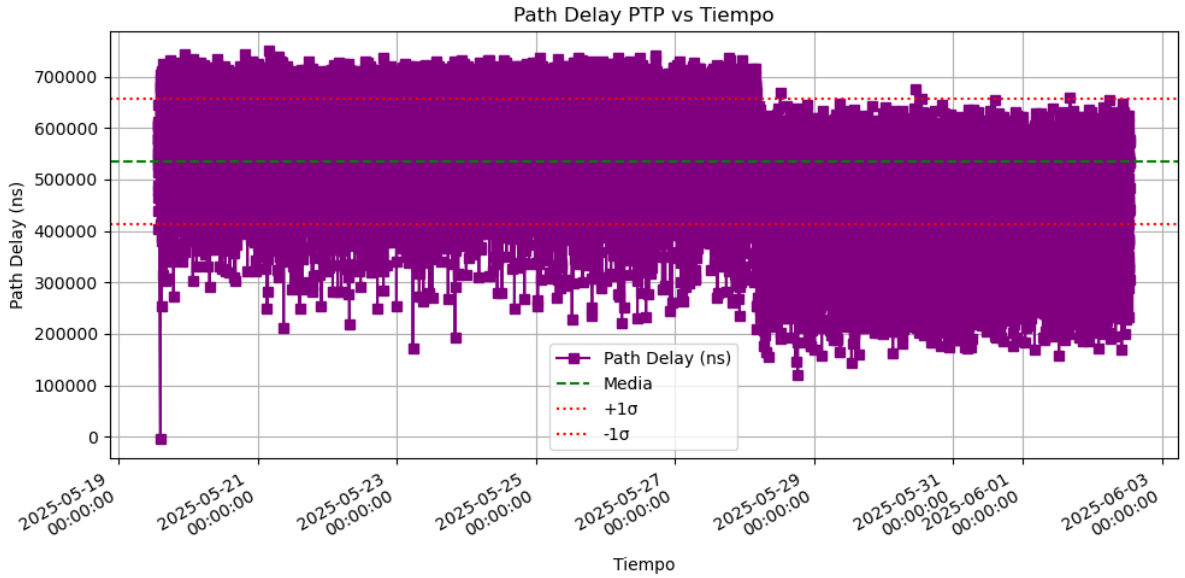

Figura 6.27: Path Delay Nodo 1: Comportamiento inestable con un pico inicial superior a 3.5 millones de nanosegundos. Tras el arranque, el *delay* converge a valores más estables, reflejando una red virtual con latencias variables.

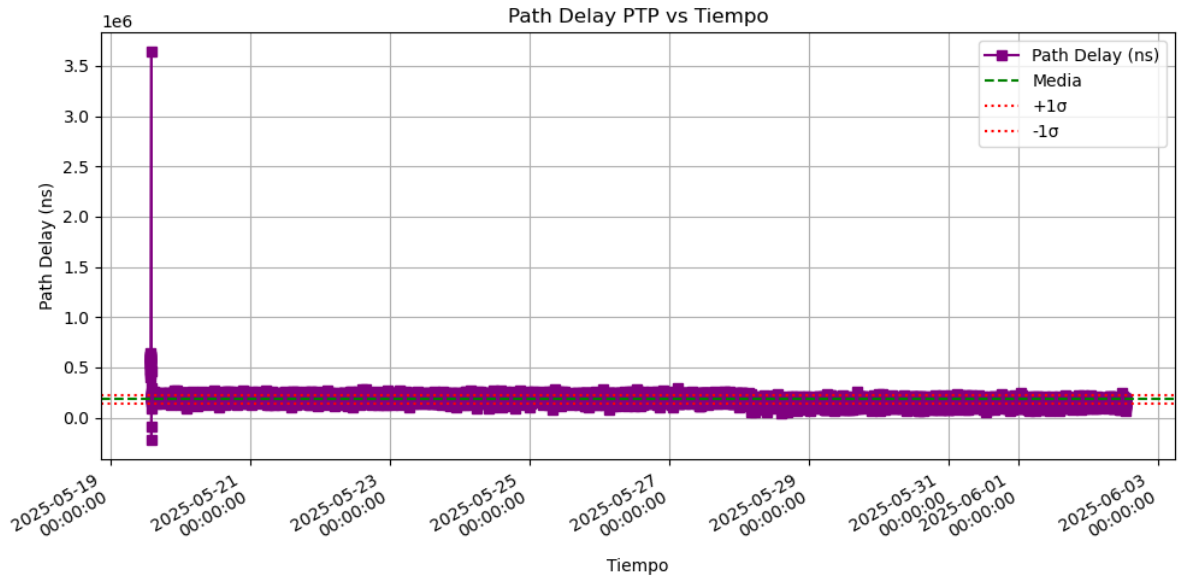

Figura 6.28: Path Delay Nodo 2: *Delay* contenido y estable en torno a la media. Cliente colocalizado con el maestro, lo que se traduce en una latencia de red baja y consistente.

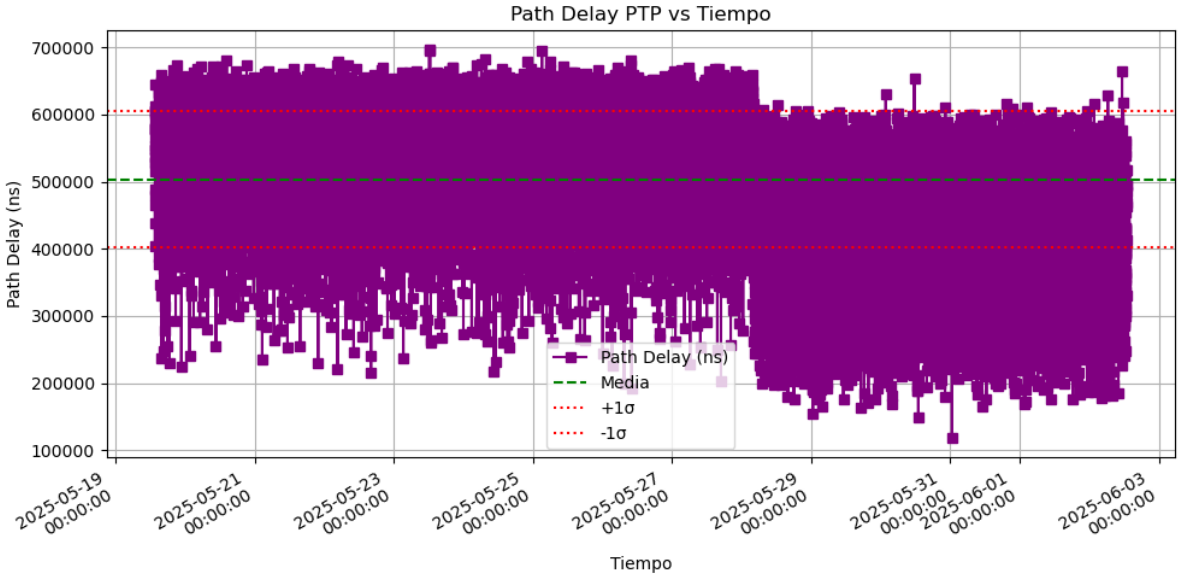

Figura 6.29: Path Delay Nodo 3: Distribución general estable con mayor dispersión y presencia de valores atípicos por debajo del umbral esperado. Posible impacto de *jitter* o carga variable en el host.

6.4. Comparativa de *White Rabbit*

No ha sido posible, al final, recolectar métricas con los dispositivos *White Rabbit* debido a la falta de un cable coaxial para conectar la señal del *Secure Sync 2400* al *switch* y, de esta manera, poder disciplinarlo.

Capítulo 7

Impacto del trabajo

A continuación, se explicará el impacto general del trabajo y los Objetivos de Desarrollo Sostenible que enlazan con el trabajo realizado.

7.1. Impacto general

Las redes de sincronía temporal representan un pilar fundamental para el desarrollo e investigación de infraestructuras cuánticas distribuidas. A diferencia de las redes clásicas, donde la sincronización temporal solo mejora el rendimiento, en las redes cuánticas es un requisito **crítico** para la correcta gestión de los diferentes dispositivos involucrados en la red.

En este contexto, una infraestructura robusta de sincronización temporal permite garantizar la coordinación precisa entre los distintos sistemas distribuidos que intervienen en la gestión y distribución de claves, como los servidores de control, los sistemas de gestión de claves (KMS) o los módulos de monitorización y seguridad. Esta sincronización resulta clave para mantener la coherencia entre nodos, evitar discrepancias en el registro de eventos y permitir el correcto funcionamiento de los protocolos de conmutación, autenticación y gestión de sesiones seguras. Además, facilita la escalabilidad hacia arquitecturas multinodo, condición indispensable para el futuro desarrollo de la ***quantum internet***.

Por tanto, los avances en sincronización temporal no solo mejoran la eficiencia de las redes cuánticas actuales, sino que constituyen una infraestructura habilitadora clave para su adopción a gran escala en el ámbito científico, industrial y estratégico.

7.2. Objetivos de Desarrollo Sostenible

A continuación, se explicarán los objetivos que se alinean con varios Objetivos de Desarrollo Sostenible (ODS) definidos por Naciones Unidas en la Agenda 2030. En particular, contribuye a los siguientes, mostrados en la figura 7.1.

- **ODS 9 – Industria, innovación e infraestructura:** El desarrollo de infraestructuras de red de alta precisión y resiliencia, como las que permiten sincronización temporal avanzada, puede servir para sostener ecosistemas tecnológicos

Figura 7.1: Representación de los 17 Objetivos de Desarrollo Sostenible establecidos por la ONU en la Agenda 2030, que promueven el desarrollo económico, social y ambiental a nivel global.

seguros y modernos. Este trabajo aporta directamente al diseño de infraestructuras digitales más robustas, preparadas para integrar sistemas distribuidos y críticos como los relacionados con la ciberseguridad, la gestión de claves o las comunicaciones sensibles.

- **ODS 11 – Ciudades y comunidades sostenibles:** El despliegue y correcto funcionamiento de la red MadQCI supone un paso estratégico hacia la digitalización segura y eficiente de los entornos urbanos. Una infraestructura de comunicaciones avanzada, respaldada por mecanismos de sincronización temporal precisos, permite coordinar con fiabilidad servicios críticos para las ciudades inteligentes, como la gestión de infraestructuras públicas o la seguridad urbana.
- **ODS 16 – Paz, justicia e instituciones sólidas:** La fiabilidad en la distribución de tiempos es un componente esencial para la trazabilidad, la verificación de eventos y la seguridad de la información. Sistemas de sincronización precisos refuerzan la confianza en los entornos digitales, particularmente en aquellos donde es necesario auditar, autenticar o coordinar decisiones en red. Esto contribuye al fortalecimiento de instituciones digitales transparentes y seguras.
- **ODS 17 – Alianzas para lograr los objetivos:** El enfoque colaborativo en redes como MadQCI, que agrupan nodos de distintas instituciones (universidades,

centros de investigación, entidades públicas y privadas), fomenta la cooperación técnica e institucional. Este trabajo sienta las bases para una interoperabilidad eficaz entre diferentes actores, apoyando el desarrollo de infraestructuras compartidas con alto valor estratégico.

Capítulo 8

Resultados y conclusiones

En este capítulo se va a explicar qué se ha conseguido en este TFG, los trabajos futuros que se tendrían que realizar y las conclusiones personales sobre el trabajo realizado.

8.1. Resultados

De los objetivos propuestos en el apartado 1.3, se han logrado los siguientes:

- **Estudiar los fundamentos teóricos necesarios para comprender la sincronización de tiempos en redes:** se ha realizado un amplio estudio de las tecnologías aplicadas en el mercado para la sincronización de tiempos, tanto en protocolos clásicos (NTP, PTP) como en protocolos avanzados. No ha sido posible realizar un estudio real con el protocolo *White Rabbit*, pero sí ha sido posible entender el funcionamiento del mismo. El capítulo 2 logra cumplir este objetivo de forma exitosa.
- **Analizar el funcionamiento de la red MadQCI:** se ha conseguido identificar los dispositivos de la red y sus necesidades de sincronía. Las redes cuánticas en la actualidad están también formadas por dispositivos clásicos y software específico que aprovecha las cualidades de este tipo de redes para acciones relativas a certificación y seguridad. Este objetivo ha sido alcanzado satisfactoriamente en el capítulo 3.
- **Diseñar una arquitectura distribuida para la gestión:** se han propuesto distintas posibilidades de arquitectura para poder cumplir los requisitos de sincronía de la red. En función de la tecnología y de la disposición de los nodos, se ha aplicado un tipo de estructura coherente para garantizar que la entrega de tiempos a las distintas máquinas mantenga la señal temporal común más próxima al reloj **maestro**. Este objetivo ha quedado parcialmente cumplido en el capítulo 4 y dependerá de la topología que asuma la red en los próximos años.
- **Integrar distintas tecnologías de sincronización en la arquitectura propuesta:** se han propuesto distintos tipos de servicio para que los diferentes investigadores que aterricen en la red tengan posibilidad de gestionar sus marcas temporales de la manera más adecuada a su experimento. Existen opciones distribuidas de entrega de servicio, opciones locales e incluso opciones para ga-

rantizar que el despliegue de las mismas sea coherente con el sistema. Se puede considerar que este objetivo se ha cumplido en el capítulo 5 con éxito.

- **Evaluar el sistema propuesto a través de métricas relevantes, simulaciones o pruebas reales:** se han realizado diferentes métricas en entornos simulados con datos reales de cómo funcionaría la sincronía dentro de los dispositivos internos de la red. Es cierto que, al no haber podido realizar la conectividad de los dispositivos *White Rabbit* y al no haber sido posible hacer un despliegue en todos los nodos de la red, este objetivo solo se ha cumplido parcialmente dentro del capítulo 6.

8.2. Conclusiones personales

A título personal, he aprendido sobre el funcionamiento de las redes cuánticas, los dispositivos involucrados en ellas y cómo llevar a cabo su sincronización temporal. Al comenzar este trabajo, mi experiencia en entornos reales de despliegue era limitada (hasta entonces solo había trabajado en entornos *cloud*), y apenas tenía conocimientos sobre los protocolos que garantizan la sincronía horaria dentro de una red. Considero especialmente valioso haber podido trabajar en un entorno real con *hardware* de calidad, así como recibir formación sobre los dispositivos cuánticos que formarán parte de esta red en el futuro. Además, este trabajo me ha permitido conocer de primera mano cómo se opera en redes locales y cómo diseñar sistemas capaces de prestar servicio a otros usuarios.

8.3. Trabajo futuro

En este trabajo han quedado tareas pendientes como la posibilidad de establecer protocolos avanzados como *White Rabbit* y la posibilidad de comprobar el funcionamiento de estas tecnologías en diversos nodos de la red.

De igual modo, es interesante la creación de una interfaz software que permita actualizar de manera automática la topología de todos los nodos NTP de la red.

También es interesante el estudio sobre cuál es la mejor manera de entregar el servicio de sincronía temporal al usuario, para poder garantizar que los investigadores interesados en trabajar dentro de la red MadQCI pueden obtener marcas temporales con la mejor precisión de la manera más sencilla posible.

Además se podría generar un archivo de infraestructura como código para poder realizar la instalación de nuevas máquinas con acceso al sistema temporal, facilitando aún más su despliegue.

Por último, sería posible crear un sistema de *Watch Dog* que pueda verificar el correcto funcionamiento de todos los sistemas, evitando así caídas dentro de la máquina virtual y garantizando que los sistemas están siempre conectados a un reloj válido.

Bibliografía

[1] Whitfield Diffie y Martin E. Hellman. «New Directions in Cryptography». En: *IEEE Transactions on Information Theory* 22.6 (1976), págs. 644-654. DOI: 10.1109/TIT.1976.1055638.

[2] Ronald L. Rivest, Adi Shamir y Leonard Adleman. «A Method for Obtaining Digital Signatures and Public-Key Cryptosystems». En: *Communications of the ACM* 21.2 (1978), págs. 120-126. DOI: 10.1145/359340.359342.

[3] Peter W. Shor. «Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer». En: *SIAM Journal on Computing* 26.5 (1997), págs. 1484-1509. DOI: 10.1137/S0097539795293172.

[4] V. Martin et al. «MadQCI: a heterogeneous and scalable SDN-QKD network deployed in production facilities». En: *npj Quantum Information* 10.1 (2024), pág. 80. DOI: 10.1038/s41534-024-00873-2. URL: https://www.nature.com/articles/s41534-024-00873-2.

[5] Leslie Lamport. «Time, clocks, and the ordering of events in a distributed system». En: *Communications of the ACM* 21.7 (1978), págs. 558-565.

[6] Flaviu Cristian. «Probabilistic clock synchronization». En: *Distributed Computing* 3.3 (1989), págs. 146-158.

[7] Riccardo Gusella y Stefano Zatti. «Clock synchronization with bounded clock drift». En: *ACM Transactions on Computer Systems (TOCS)* 7.4 (1989), págs. 376-405.

[8] David L. Mills. «Internet time synchronization: the Network Time Protocol». En: *IEEE Transactions on Communications* 33.10 (1985), págs. 1135-1143.

[9] IEEE Instrumentation and Measurement Society. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. Publicado el 8 de noviembre de 2002. IEEE, nov. de 2002.

[10] Hirschmann. *Precision Clock Synchronization – The Standard IEEE 1588*. Inf. téc. White Paper. Hirschmann, A Belden Brand, 2008. URL: https://www.industrialnetworking.com/pdf/Hirschmann_IEEE_1588.pdf.

[11] Safran Electronics & Defense. *SecureSync 2400 User Manual*. Document Part No.: 2400-5000-0050, Revision: 8.1. Safran Navigation & Timing. 2024. URL: https://safran-navigation-timing.com/wp-content/uploads/2024/11/SecureSync2400_UserManual_PN_2400-5000-0050_r8.1.pdf.

[12] Pedro Moreira et al. «White rabbit: Sub-nanosecond timing distribution over ethernet». En: *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. 2009, págs. 1-5. DOI: 10.1109/ISPCS.2009.5340196.

[13] Charles H. Bennett y Gilles Brassard. *Quantum cryptography: Public key distribution and coin tossing*. Originally presented at IEEE International Conference on Computers, Systems, and Signal Processing, Bangalore, India, 1984. 2014.

DOI: 10.48550/arXiv.2003.06557. arXiv: arXiv:2003.06557 [quant-ph]. URL: https://doi.org/10.48550/arXiv.2003.06557.

[14] J. S. Bell. «On the Einstein Podolsky Rosen Paradox». En: *Physics Physique Fizika* 1.3 (1964), págs. 195-200. DOI: 10.1103/PhysicsPhysiqueFizika.1.195.

[15] chrony project. *Chrony: A versatile implementation of the Network Time Protocol (NTP)*. https://chrony.tuxfamily.org. Acceso el 15 de abril de 2025. 2024.

[16] Richard Cochran. *linuxptp - Precision Time Protocol (PTP, IEEE 1588) implementation for Linux*. https://linuxptp.nwtime.org/. Acceso el 15 de abril de 2025. 2024.

[17] Avi Kivity et al. «KVM: the Linux Virtual Machine Monitor». En: *Proceedings of the Ottawa Linux Symposium (OLS)*. Citeseer. 2007, págs. 225-230.

[18] Fabrice Bellard. «QEMU, a fast and portable dynamic translator». En: *Proceedings of the USENIX Annual Technical Conference, FREENIX Track*. 2005, págs. 41-46.

[19] David L. Mills. «Internet Time Synchronization: The Network Time Protocol». En: *IEEE Transactions on Communications* 39.10 (1991), págs. 1482-1493. DOI: 10.1109/26.103043. URL: https://ieeexplore.ieee.org/document/103043.

[20] Safran Electronics & Defense. *White Rabbit Switch – Low Jitter (WRS-LJ)*. https://safran-navigation-timing.com/product/white-rabbit-switch-low-jitter/. Switch de 18 puertos con precisión subnanosegundo para distribución de tiempo y frecuencia. 2023.

[21] Proxmox Server Solutions GmbH. *Virtual Machines - Proxmox VE Administration Guide*. Accessed: 2025-05-30. 2024. URL: https://pve.proxmox.com/pve-docs/chapter-qm.html.

[22] Michael Kerrisk. *ioctl(2) - Linux manual page*. Accessed: 2025-05-31. 2025. URL: https://man7.org/linux/man-pages/man2/ioctl.2.html.

Apéndice A

Anexo

A continuación se adjuntaran algunos ejemplos de los datos obtenidos en formato csv y de los programas de python utilizados para el analisis de los mismos

```
timestamp,offset_s,frequency_ppm
2025-05-14T13:17:54.185512,1.3373e-05,1.08
2025-05-14T13:17:55.188392,1.3332e-05,1.08
2025-05-14T13:17:56.191155,1.3291e-05,1.08
2025-05-14T13:17:57.194184,1.325e-05,1.08
2025-05-14T13:17:58.197075,1.321e-05,1.08
2025-05-14T13:17:59.199630,1.3169e-05,1.08
2025-05-14T13:18:00.203378,1.3128e-05,1.08
2025-05-14T13:18:01.206418,1.3087e-05,1.08
2025-05-14T13:18:02.208870,1.3046e-05,1.08
2025-05-14T13:18:03.211723,1.3006e-05,1.08
2025-05-14T13:18:04.214425,1.2965e-05,1.08
2025-05-14T13:18:05.218194,1.2924e-05,1.08
2025-05-14T13:18:06.221244,1.2883e-05,1.08
2025-05-14T13:18:07.223724,1.2842e-05,1.08
2025-05-14T13:18:08.226757,1.2802e-05,1.08
2025-05-14T13:18:09.229519,1.2761e-05,1.08
2025-05-14T13:18:10.233375,1.272e-05,1.08
2025-05-14T13:18:11.236289,1.2679e-05,1.08
2025-05-14T13:18:12.239078,1.2638e-05,1.08
2025-05-14T13:18:13.242110,1.2597e-05,1.08
2025-05-14T13:18:14.245210,1.2557e-05,1.08
2025-05-14T13:18:15.249110,1.2516e-05,1.08
2025-05-14T13:18:16.251770,1.2475e-05,1.08
2025-05-14T13:18:17.254510,1.2434e-05,1.08
2025-05-14T13:18:18.257507,1.2393e-05,1.08
2025-05-14T13:18:19.260309,1.2353e-05,1.08
2025-05-14T13:18:20.264236,1.2312e-05,1.08
2025-05-14T13:18:21.267211,1.2271e-05,1.08
```

Listing A.1: Ejemplo de métricas obtenidas en formato .csv en el primer nodo para la realización de las métricas de resultado (las métricas tomadas llegaron a 167270)

```
import pandas as pd
import matplotlib.pyplot as plt
```

```
import matplotlib.dates as mdates

# Cargar CSV
df = pd.read_csv('chrony_metrics_random.csv', parse_dates=['timestamp'])
df['offset_s'] = df['offset_s'].astype(float)
df['frequency_ppm'] = df['frequency_ppm'].astype(float)

# Estadsticas
offset_mean = df['offset_s'].mean()
offset_std = df['offset_s'].std()
freq_mean = df['frequency_ppm'].mean()
freq_std = df['frequency_ppm'].std()

print("Estadsticas del offset (s):")
print(df['offset_s'].describe())
print("\nEstadsticas de la frecuencia (ppm):")
print(df['frequency_ppm'].describe())

# ---------- OFFSET ----------
plt.figure(figsize=(10, 5))
plt.plot(df['timestamp'], df['offset_s'], marker='o', linestyle='-', label=
    'Offset (s)')
plt.axhline(offset_mean, color='green', linestyle='--', label='Media')
plt.axhline(offset_mean + offset_std, color='red', linestyle=':', label='+1
    ')
plt.axhline(offset_mean - offset_std, color='red', linestyle=':', label='-1
    ')
plt.title('Offset NTP vs Tiempo')
plt.xlabel('Tiempo')
plt.ylabel('Offset (s)')
plt.legend()
plt.grid(True)

# Formato de fechas legible
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d\n%H:%M:%
    S'))
plt.gcf().autofmt_xdate()

plt.tight_layout()
plt.savefig("offset_ntp.png")
plt.show()

# ---------- FRECUENCIA ----------
plt.figure(figsize=(10, 5))
plt.plot(df['timestamp'], df['frequency_ppm'], marker='x', linestyle='-',
    color='orange', label='Frecuencia (ppm)')
plt.axhline(freq_mean, color='green', linestyle='--', label='Media')
plt.axhline(freq_mean + freq_std, color='red', linestyle=':', label='+1')
plt.axhline(freq_mean - freq_std, color='red', linestyle=':', label='-1')
plt.title('Frecuencia NTP vs Tiempo')
plt.xlabel('Tiempo')
plt.ylabel('Frecuencia (ppm)')
plt.legend()
plt.grid(True)

plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d\n%H:%M:%
```

```
    S'))
plt.gcf().autofmt_xdate()

plt.tight_layout()
plt.savefig("frecuencia_ntp.png")
plt.show()
```

Listing A.2: Script en Python para análisis de métricas NTP con visualización de offset (en segundos) y frecuencia

```
timestamp,offset_ns,estado,freq,path_delay_ns,grandmaster_mac
2025-05-19T14:23:57.840256,,s0,0,-190617,bc:24:11:ff:fe:a2
2025-05-19T14:23:58.843203,,s0,0,-190617,bc:24:11:ff:fe:a2
2025-05-19T14:23:59.845863,,s0,0,114872,bc:24:11:ff:fe:a2
2025-05-19T14:24:00.848244,,s0,0,114872,bc:24:11:ff:fe:a2
2025-05-19T14:24:01.850385,,s0,0,313861,bc:24:11:ff:fe:a2
2025-05-19T14:24:02.852188,,s0,0,313861,bc:24:11:ff:fe:a2
2025-05-19T14:24:03.853489,,s0,0,313861,bc:24:11:ff:fe:a2
2025-05-19T14:24:04.855296,,s0,0,313861,bc:24:11:ff:fe:a2
2025-05-19T14:24:05.856407,,s0,0,299593,bc:24:11:ff:fe:a2
2025-05-19T14:24:06.857369,,s0,0,299593,bc:24:11:ff:fe:a2
2025-05-19T14:24:07.858310,,s0,0,332803,bc:24:11:ff:fe:a2
2025-05-19T14:24:08.859305,,s0,0,332803,bc:24:11:ff:fe:a2
2025-05-19T14:24:09.860341,,s0,0,400934,bc:24:11:ff:fe:a2
2025-05-19T14:24:10.861177,,s0,0,450721,bc:24:11:ff:fe:a2
2025-05-19T14:24:11.861983,,s0,0,450721,bc:24:11:ff:fe:a2
2025-05-19T14:24:12.862765,,s0,0,450721,bc:24:11:ff:fe:a2
2025-05-19T14:24:13.868844,-44.713623839184365,s1,1089782,498313,bc:24:11:
    ff:fe:a2
2025-05-19T14:24:14.866758,-11.722965829925643,s2,959539,498313,bc:24:11:ff
    :fe:a2
2025-05-19T14:24:15.865981,-27.95041800263849,s2,844951,498313,bc:24:11:ff:
    fe:a2
2025-05-19T14:24:16.864682,24.602063823866715,s2,696816,567847,bc:24:11:ff:
    fe:a2
2025-05-19T14:24:17.864492,2.0716069344772583,s2,639647,567847,bc:24:11:ff:
    fe:a2
2025-05-19T14:24:18.864052,6.015773197736053,s2,554662,624395,bc:24:11:ff:
    fe:a2
2025-05-19T14:24:19.863543,-49.52906191089649,s2,462372,774719,bc:24:11:ff:
    fe:a2
2025-05-19T14:24:20.862882,5.1482230028768425,s2,370289,863333,bc:24:11:ff:
    fe:a2
```

Listing A.3: Ejemplo de métricas obtenidas en formato .csv en el primer nodo para la realización de las métricas de resultado (las metricas tomadas llegaron a 1203743)

```
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates

# ---------- CARGA Y LIMPIEZA DE DATOS ----------

# Cargar CSV
```

```
df = pd.read_csv('ptp_log.csv')

# Convertir 'timestamp' a datetime, eliminando cualquier fila inv lida
df['timestamp'] = pd.to_datetime(df['timestamp'], errors='coerce')
df = df.dropna(subset=['timestamp'])

# Convertir columnas num ricas
df['offset_ns'] = pd.to_numeric(df['offset_ns'], errors='coerce')
df['freq'] = pd.to_numeric(df['freq'], errors='coerce')
df['path_delay_ns'] = pd.to_numeric(df['path_delay_ns'], errors='coerce')

# ---------- ESTAD STICAS ----------

offset_mean = df['offset_ns'].mean()
offset_std = df['offset_ns'].std()
freq_mean = df['freq'].mean()
freq_std = df['freq'].std()
delay_mean = df['path_delay_ns'].mean()
delay_std = df['path_delay_ns'].std()

print("Estad sticas del offset (ns):")
print(df['offset_ns'].describe())
print("\nEstad sticas de la frecuencia:")
print(df['freq'].describe())
print("\nEstad sticas del path delay (ns):")
print(df['path_delay_ns'].describe())

# ---------- OFFSET ----------
plt.figure(figsize=(10, 5))
plt.plot(df['timestamp'], df['offset_ns'], marker='o', linestyle='-', label
    ='Offset (ns)')
plt.axhline(offset_mean, color='green', linestyle='--', label='Media')
plt.axhline(offset_mean + offset_std, color='red', linestyle=':', label='+1
    ')
plt.axhline(offset_mean - offset_std, color='red', linestyle=':', label='-1
    ')
plt.title('Offset PTP vs Tiempo')
plt.xlabel('Tiempo')
plt.ylabel('Offset (ns)')
plt.legend()
plt.grid(True)
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m-%d\n%H:%M:%
    S'))
plt.gcf().autofmt_xdate()
plt.tight_layout()
plt.savefig("offset_ptp.png")
plt.show()
```

Listing A.4: Script en Python para análisis de logs PTP: offset, frecuencia y retardo

Este documento esta firmado por

	**Firmante**	CN=tfgm.fi.upm.es, OU=CCFI, O=ETS Ingenieros Informaticos - UPM, C=ES
	Fecha/Hora	Wed Jun 04 15:27:41 CEST 2025
	Emisor del Certificado	EMAILADDRESS=camanager@etsiinf.upm.es, CN=CA ETS Ingenieros Informaticos, O=ETS Ingenieros Informaticos - UPM, C=ES
	Numero de Serie	561
	Metodo	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signature)