Universidad de Valladolid

Máster universitario

Ingeniería Informática

Trabajo Fin de Máster

Despliegue automatizado y evaluación comparativa de nodos Bitcoin en Linux adaptados a perfiles de usuario

Realizado por **José Ulloa Araya**

✠✠✠

Universidad de Valladolid

8 de noviembre de 2025

Tutor: Diego Llanos

Universidad de Valladolid

Máster universitario en Ingeniería Informática

D. Diego Llanos, profesor del departamento de Informática, área de Arquitectura y Tecnología de Computadores.

Expone:

Que el alumno D. José Ulloa Araya, ha realizado el Trabajo final de Máster en Ingeniería Informática titulado "Despliegue automatizado y evaluación comparativa de nodos Bitcoin en Linux adaptados a perfiles de usuario".

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Valladolid, 8 de noviembre de 2025

Vº. Bº. del Tutor:

D. nombre tutor

Vº. Bº. del co-tutor:

D. nombre co-tutor

Resumen

El Trabajo Fin de Máster tiene como objetivo el diseño y desarrollo de un sistema automatizado para la instalación y configuración de nodos Bitcoin sobre sistemas Linux, adaptado a las necesidades específicas de distintos perfiles de usuario. En particular tienen cabida perfiles como son el de validadores, analistas de datos, desarrolladores de wallets, operadores de nodos Lightning y usuarios con fines educativos. El proyecto consiste en la puesta en marcha de un script que despliega un nodo Bitcoin Core con parámetros específicos y la evaluación rigurosa del efecto de distintas configuraciones en el rendimiento del sistema. Se han registrado distintas métricas como el uso de la CPU, la RAM, el disco y el ancho de banda durante la sincronización de la blockchain. El objetivo principal es proporcionar una herramienta que facilite el despliegue de nodos, junto a documentar el comportamiento del software Bitcoin Core ante distintos escenarios de uso. Los resultados obtenidos permiten identificar configuraciones óptimas según el perfil de cada usuario.

Descriptores

Bitcoin, nodos completos, automatización, monitorización de métricas, perfiles de usuario.

Abstract

The Master's Thesis aims to design and develop an automated system for installing and configuring Bitcoin nodes on Linux systems, tailored to the specific needs of different user profiles. Particularly suitable for users are validators, data analysts, wallet developers, Lightning node operators, and educational users. The project consists of implementing a script that deploys a Bitcoin Core node with specific parameters and rigorously evaluating the effect of different configurations on system performance. Various metrics such as CPU, RAM, disk, and bandwidth usage were recorded during blockchain synchronization. The main objective is to provide a tool that facilitates node deployment, while also documenting the behavior of the Bitcoin Core software in different usage scenarios. The results obtained allow for identifying optimal configurations based on each user's profile.

Keywords

Bitcoin, full nodes, automation, metrics monitoring, user profiles.

Índice general

Índice de figuras

Índice de tablas

1: Introducción

1.1. ¿Qué es Bitcoin y cómo funciona?

Bitcoin es un tipo de sistema monetario digital descentralizado que fue propuesto por Satoshi Nakamoto en el 2008 con la finalidad de permitir el intercambio de valor entre pares sin necesidad de utilizar intermediarios (bancos, entidades financieras, etc.) [8]. Su funcionamiento tiene como base una red P2P (*peer-to-peer*), donde entonces, todos los participantes colaboran en la validación, la propagación y el almacenamiento de las transacciones.

A diferencia de los sistemas tradicionales, Bitcoin carece de una autoridad central que se encargue de emitir o validar dinero. Más bien emplea un mecanismo de consenso distribuido entre los nodos de la red para validar la legitimidad de las transacciones, en donde cada nodo de la red posee una copia del historial de transacciones, ya sea de manera completa o parcial, de modo que se pueda asegurar así la transparencia, resistencia a la censura y la seguridad criptográfica [5].

En la Figura 1.1 se observa cómo interactúan los distintos elementos que componen el ecosistema Bitcoin, pero desde una manera general, de modo que sea fácilmente entendible. La estructura es fácil, pues los usuarios gestionan sus claves privadas mediante una *wallet*, por otro lado, los mineros verifican y añaden bloques a la cadena, mientras que los nodos de red se encargan de la propagación y sincronización de la información, de tal forma que los nodos son los que permiten mantener la integridad del sistema sin la necesidad de una autoridad central.

Figura 1.1: Esquema general del ecosistema Bitcoin. Fuente: *Mastering Bitcoin* [5].

Bitcoin, en definitiva, mezcla tres elementos: la base de datos pública que es la *blockchain*) (o cadena de bloques), el protocolo de consenso distribuido y un sistema de criptografía basado en las claves públicas/privadas. A través de estos mecanismos, los usuarios pueden realizar transferencias de bitcoins sin tener que confiar en un tercero, solo mediante la firma digital de las operaciones y la propagación de las mismas a la red para su validación.

En definitiva, Bitcoin es la combinación de tres elementos: la base de datos pública que representa la *blockchain* (o cadena de bloques) por el que se van encadenando todos los bloques de la base de datos entre sí, el protocolo de consenso distribuido y, por último, el esquema criptográfico basado en claves públicas/privadas. Y gracias a estos tres mecanismos, los usuarios puedan llevar a cabo transferencias de bitcoins sin la necesidad de confiar en terceros, pues solo se requiere la firma digital de las transferencias y la propagación de sus transacciones a la red en el procedimiento de validación.

1.2. Bloques, transacciones y consenso

Uno de los aspectos más relevantes de Bitcoin se encuentra en su arquitectura de bloques, de la misma manera que el proceso de consenso que es necesario para poder enlazarlos uno a uno y en un orden cronológico absoluto e inalterable en el que cada bloque de Bitcoin contiene un conjunto de las transacciones que han de ser contrastadas, un sello de tiempo (*timestamp*) y una referencia criptográfica del bloque anterior. Todo ello da como resultado una cadena, una secuencialidad de bloques, a la que se le ha dado el nombre de *blockchain* [10].

Por su parte las transacciones constituyen el centro operativo de la red, pues cada una de ellas permite registrar el movimiento de bitcoins desde una determinada dirección de origen asociada al registro de la transacción, a una o más direcciones de destino. Las transacciones son firmadas digitalmente por el propietario de los fondos y son verificadas por los nodos antes de ser incorporadas a un bloque.

Como muestra la Figura 1.2, cada bloque está compuesto de una estructura determinada que incluye un encabezado que guarda el *hash* del bloque anterior, el timestamp o sello de tiempo, la raíz de Merkle[1] que permite agrupar las transacciones, entre otros datos. Este encadenamiento criptográfico es precisamente lo que asegura que la modificación de un bloque haga que todos los demás bloques que le siguen se invaliden, lo que garantiza la inmutabilidad del registro.

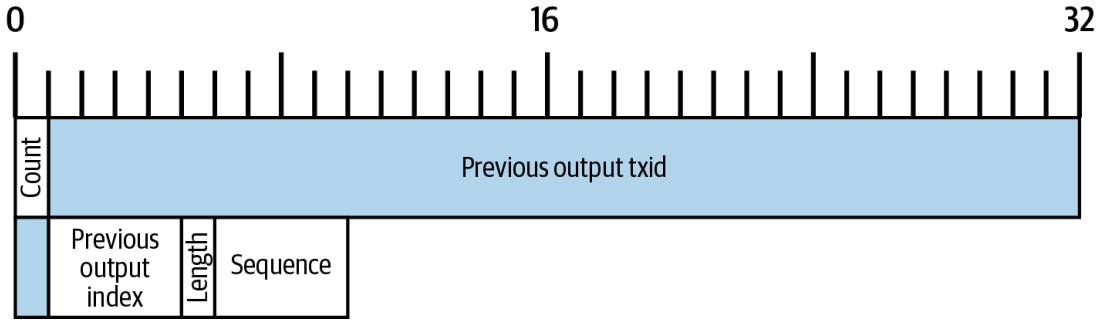

Figura 1.2: Estructura de un bloque Bitcoin, mostrando su encabezado y lista de transacciones. Fuente: *Mastering Bitcoin* [5].

En Bitcoin, el consenso es alcanzado por medio del algoritmo *Proof of Work* (PoW), el cual requiere que los mineros gasten potencia computacional que está destinada a la resolución de algún problema criptográfico determinado. Este método asegura que el añadir nuevos bloques en una cadena de bloques se vuelva muy costoso en virtud de las energías consumidas (tanto las energías eléctricas como el tiempo), favoreciendo de este modo que resulte difícil cualquier intento de manipulación de la cadena. Antonopoulos [5] hace hincapié en que este proceso convierte a la blockchain en un registro de acceso inmodificable que puede ser verificado por cualquier participante de la red.

1.3. Mineros y nodos: roles y diferencias

En la red Bitcoin podemos encontrar a diferentes tipos de participantes que ocupan cada uno una función propia dentro de esta red. Por ejemplo, los **nodos** son los encargados de ir conservando una copia del historial completo de la blockchain y de verificar las transacciones y los bloques en función a las reglas del protocolo. En cambio, los **mineros**,

[1] El "árbol de Merkle.es una estructura de datos binaria propuesta por Ralph Merkle (1980) y permite la verificación de grandes conjuntos de datos a través de un hash raíz (*Merkle Root*) calculado a partir de las transacciones individuales de un bloque en concreto.

son nodos especializados que compiten entre ellos para ser el que consigue crear bloques mediante la resolución del problema de PoW.

Es cierto que todos los mineros son nodos, pero no todos los nodos son mineros, dado que los nodos completos (*full nodes*) desempeñan un papel en la verificación independiente de las transacciones, pues no validarán ningún bloque inválido incluso si lo envía un minero con gran capacidad computacional [5], con lo que se garantiza que el poder de decisión no dependa totalmente de las personas que más recursos tienen, sino de la conformidad con las reglas que establece el protocolo.

La relación entre usuarios, nodos y mineros queda reflejada de manera simplificada en la Figura 1.3. Los usuarios generan las transacciones firmadas digitalmente con sus claves privadas; los nodos validan y propagan las transacciones; los mineros agrupan las transacciones en bloques y aseguran el mantenimiento de bloques dentro de la cadena de transmisión mediante el mecanismo de consenso PoW.

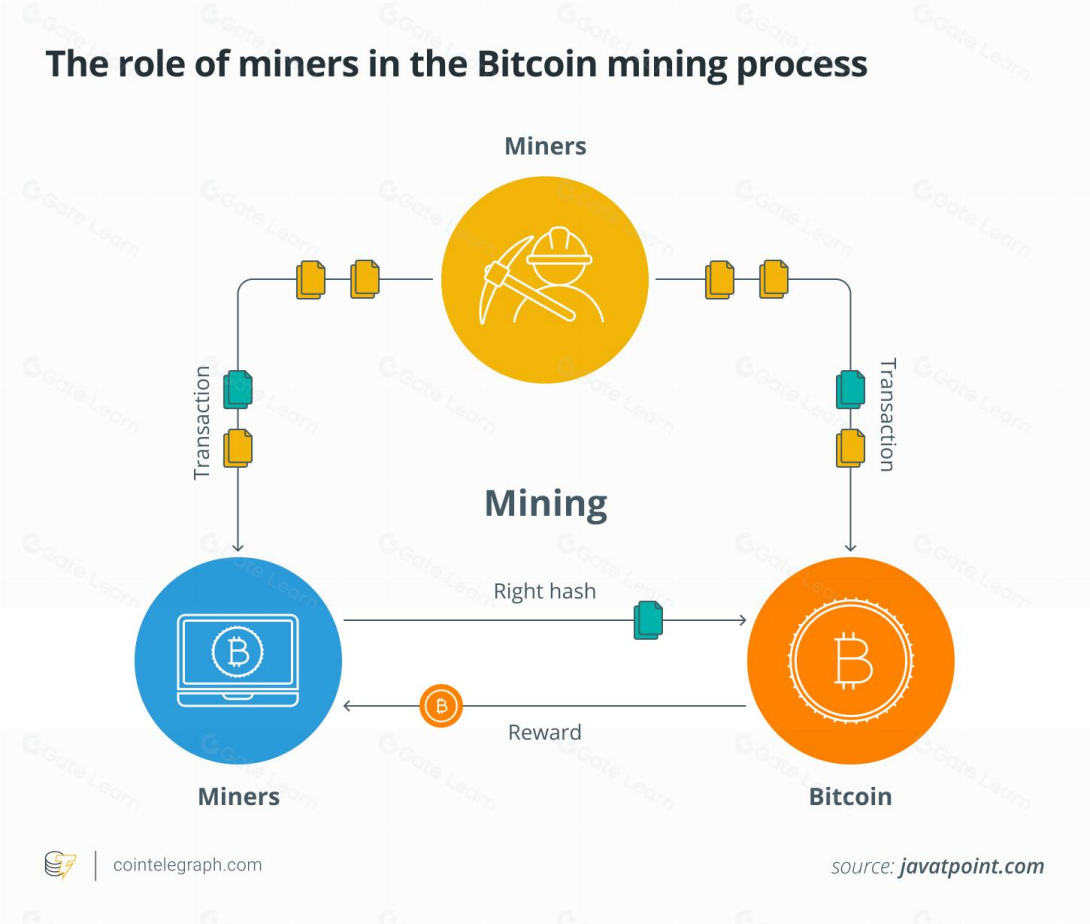

Figura 1.3: Relación entre usuarios, nodos y mineros dentro de la red Bitcoin. Fuente: *Gate.com* [13].

Los mineros, que de cierta manera están a cargo de asegurar la red, obtienen una recompensa en forma de nuevos bitcoins y comisiones por transacción, lo que claramente los induce a querer participar. Como decía Drescher y otros [10], esta economía interna garantiza que exista una compensación entre el esfuerzo computacional que se pone en juego y la estabilidad del sistema.

1.4. Estructura de la blockchain

La blockchain de Bitcoin puede considerarse como un libro mayor que se halle distribuido en los nodos existentes en la red que soporta la criptodivisa, de tal forma que un bloque se podrá entender como una de las páginas que lo componen y, a su vez, cada transacción, de alguna forma, como una de las entradas del libro. Cabe añadir que, ya desde una parte más técnica del ámbito, cada bloque dentro de la blockchain de Bitcoin se encuentra organizado principalmente en tres secciones: un bloque de encabezado (*block header*), una lista de transacciones y una referencia criptográfica a la cadena del bloque anterior *Hash* [14].

En la Figura 1.4 se puede observar la estructura jerárquica que tiene dicha blockchain, donde cada bloque tiene un encabezado que incluye el *Hash* del bloque anterior, la raíz de Merkle obtenida mediante el agrupamiento de todas las transacciones de la capa mediante el árbol binario y el nivel de dificultad del PoW, se comporta de modo que si se modifica un bloque se sabe que necesariamente se modificarán todos los bloques que sigan. De esta forma se asegura que la cadena está íntegra.

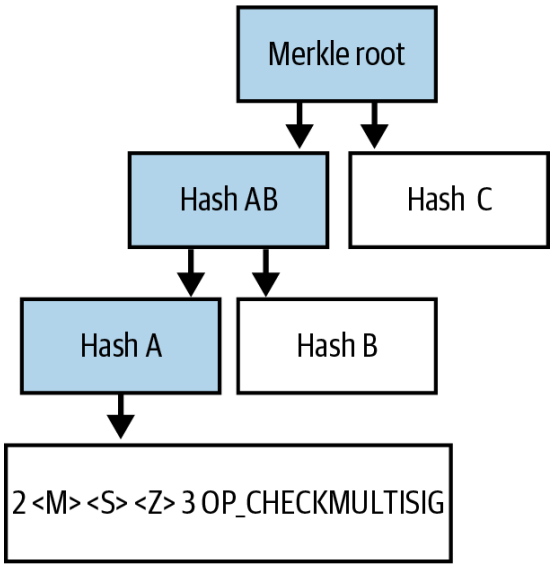

Figura 1.4: Estructura jerárquica de la blockchain de Bitcoin en donde se aprecia que la raíz de Merkle facilita el agrupamiento de los hashes de las transacciones del bloque formando un árbol binario que garantiza la integridad de las mismas. Fuente: *Mastering Bitcoin* [5].

Antonopoulos [5] justifica el uso del árbol de Merkle ya que permite reducir la complejidad de las verificaciones. Un nodo puede validar la validez de una transacción sin la necesidad de descargar toda la blockchain. Esta propiedad es la base de los clientes ligeros (*SPV nodes*), que optimizan el almacenamiento y el ancho de banda.

1.5. Relevancia de los nodos para la descentralización

Los nodos son la base de la descentralización existente en Bitcoin, en donde cada nodo completo verifica de forma autónoma que se cumplan las reglas del protocolo sin hacer uso de ningún tipo de autoridad central ni de los mineros. De esta manera, la seguridad de la red no se sustenta en la confianza, sino más bien en la verificación colectiva. [4, 1].

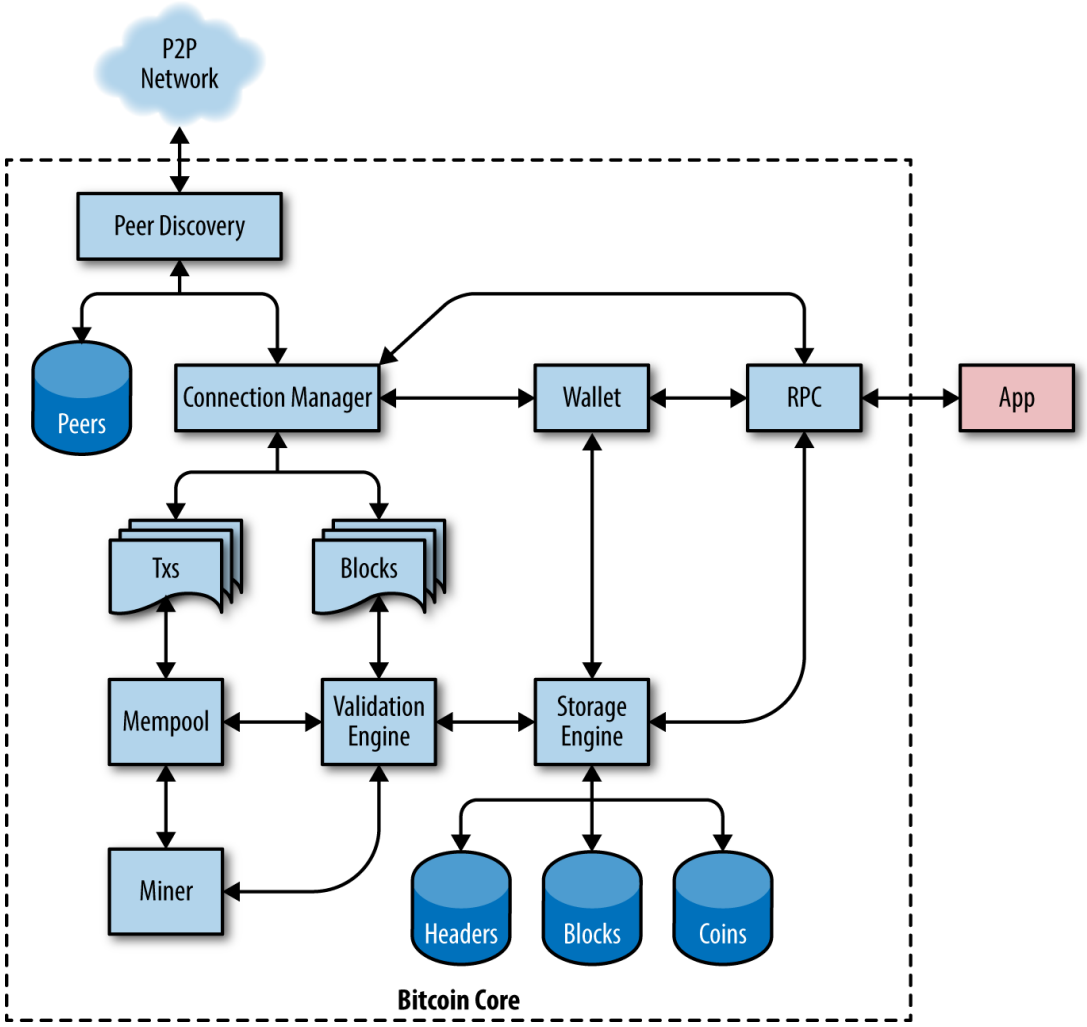

Figura 1.5: Arquitectura interna de un nodo completo ejecutando Bitcoin Core. Fuente: *Mastering Bitcoin* [5].

Tal como se observa en la Figura 1.5, la distribución global de los nodos resulta ser una variable determinante en el índice de resiliencia del sistema, en donde una red con nodos dispersos por el mundo impide que un ataque o incluso un apagón local anule el funcionamiento de Bitcoin.

La diversidad geográfica, junto a la diversidad técnica de los nodos, hace que la robustez del sistema sea aún mayor frente a la censura o ataques. Cuando un nodo (o algunos de ellos) se desconectan, el resto de la red continúa operando con normalidad. Según Champagne [8], esta fue la razón por la cual Bitcoin es “el sistema más resistente a la corrupción y la manipulación humana jamás diseñado”.

Los nodos, además, son fundamentales para la mantención de la transparencia y de la auditabilidad pública en la blockchain, dado que cualquier usuario puede instalar el software de referencia (Bitcoin Core) y validar por sí mismo todas las transacciones realizadas. De forma resumida, la idea de "no confiar, sino verificar"da cuenta de la filosofía técnica y ética de Bitcoin.

1.6. Contexto y motivación

En los apartados previos se ha expuesto el funcionamiento interno del sistema y los fundamentos teóricos del sistema de Bitcoin; partiendo de esta base, el actual trabajo presenta el despliegue automatizado y la evaluación empírica de su software de referencia: `Bitcoin Core`.

Bitcoin Core es el software de referencia para la operación de nodos en la red Bitcoin. Estos nodos son parte del funcionamiento de la red porque comprueban las transacciones, validan bloques y garantizan la descentralización del sistema. Aun así, poner en marcha un nodo completo no es algo que se pueda hacer fácilmente. Necesita unos conocimientos técnicos avanzados, consume una cantidad importante de recursos de hardware y plantea problemas de configuración en general, y todavía más en un entorno académico o de investigación donde se requiere la reproducibilidad y la comparabilidad.

El interés por el análisis empírico de las redes blockchain ha evidenciado la necesidad de herramientas ligeras y accesibles para poder desplegar automatizadamente nodos y estudiar su comportamiento, luego ya con esta idea en mente, es cuando surge la motivación de este Trabajo Fin de Máster: diseñar una metodología reproducible respaldada por un script de instalación y monitorización y capaz de evaluar el consumo de recursos del cliente Bitcoin Core en función de diferentes escenarios de uso.

1.7. Planteamiento del problema

La instalación manual de Bitcoin Core incluye una serie de pasos que difieren por plataforma, y la monitorización tradicional emplea herramientas pesadas que suponen un mayor consumo y una mayor complejidad. Por otro lado, la literatura científica sobre

rendimiento de nodos Bitcoin es en su mayoría simulaciones, escasa en estudios empíricos sobre entornos reales con hardware ajustado.

Por consiguiente, la problemática que se plantea en el presente trabajo de fin de máster es la siguiente: por un lado, simplificar el proceso de puesta en marcha del despliegue y configuración de un nodo Bitcoin completo según distintos perfiles, y por otro lado, contar con una herramienta de análisis comparativo de manera que se puedan capturar métricas relevantes (CPU, RAM, disco, red, procesos) en las fases de sincronización y post-sincronización de dichos perfiles.

1.8. Aporte del TFM

La creación de un script en Bash (`script_tfm.sh`) que permite automatizar la instalación, la configuración y la monitorización de Bitcoin Core es la contribución principal de este trabajo, además de contar con un módulo de análisis en Python (`graficos.py`) que transforma logs CSV en gráficos y tablas comparativas. La principal innovación consiste en la definición de perfiles de usuario (Validador, Analista, dApps, Lightning, Educador y Estándar) que representan distintos casos de uso y la forma de estudiarlos según su implicación en el uso de los recursos del sistema.

Este enfoque permite hacer las comparaciones de configuraciones de una forma sistemática, permitiendo encontrar un punto intermedio entre la simplicidad necesaria para el contexto docente y la necesidad técnica frente a escenarios de producción.

1.9. Metodología general

La metodología aportada consta de la automatización, la monitorización y el análisis comparativo. En primer lugar, se idearon los perfiles de usuario en función de las necesidades planteadas. En segundo lugar, se ejecutó un script que configura la compilación de Bitcoin Core, la configuración personalizada y la obtención de métricas en CSV. En tercer lugar, los datos obtenidos fueron procesados con Python para la creación de gráficas y tablas que se encargan de sintetizar las diferencias entre perfiles para el desarrollo del TFM.

Este ciclo iterativo hizo posible refinar tanto el código como los experimentos para garantizar la validez y la repetibilidad de los resultados.

1.10. Estructura del documento

El trabajo está dividido en siete capítulos. El Capítulo 1 describe el contexto del trabajo y los objetivos. El Capítulo 2 propone los objetivos generales y específicos. El Capítulo 3 explica los conceptos teóricos y técnicos de referencia. En el Capítulo 4 se presentan las herramientas utilizadas. El Capítulo 5 se centra en el desarrollo del proyecto y va explicando los resultados experimentales, las decisiones metodológicas y la forma

de sintetizar el desarrollo del mismo. El Capítulo 6 coloca el trabajo en el marco de la literatura existente. Por último, el Capítulo 7 presenta las conclusiones y las futuras líneas de investigación. Finalmente, los anexos incluyen material complementario (plan de proyecto, requisitos, diseño, documentación del programador, manual de usuario, etc.).

2: Objetivos del proyecto

2.1. Objetivo general

- Automatizar la instalación de nodos Bitcoin con la opción de elegir distintos perfiles de usuario y evaluar su rendimiento.

2.2. Objetivos específicos

- Desarrollar un script de instalación (script_tfm.sh).
- Generar perfiles funcionales con sus respectivas configuraciones en el fichero bitcoin.conf.
- Medir y registrar consumo de CPU, RAM, disco y red.
- Procesar datos relacionados con los distintos perfiles de instalación y generar gráficos para sus estudios y comparaciones dentro del TFM.
- Comparar perfiles y proponer configuraciones óptimas.

3: Conceptos teóricos

3.1. Bitcoin Core y nodos

Considerando que el objetivo del proyecto es proponer una herramienta `flexible` y `reproducible`, es necesario determinar perfiles de usuario representativos de los principales casos de uso que podrían darse en la operación de nodos Bitcoin. Esa clasificación no pretende ser exhaustiva ni rígida (como lo serían los roles de protocolo, como por ejemplo los mineros), sino `funcional`: cada perfil corresponde a un cierto número de necesidades operativas (validación, análisis, desarrollo, docencia, etc.) que orientan la parametrización del sistema y que permitirán llevar a cabo evaluaciones comparativas de rendimiento de forma realista.

Full node

Los nodos completos son aquellos que permiten almacenar la totalidad de la cadena de bloques e ir validando todas las transacciones y bloques de acuerdo con las reglas de consenso; estos nodos son, por tanto, la base de la red, dado que garantizan que el sistema funcione de manera descentralizada y sin depender de terceros. Para Antonopoulos y Harding, los nodos completos son la columna vertebral de la red Bitcoin, puesto que cada nodo completo valida de forma independiente la validez de los bloques y de las transacciones [5].

Pruned node

La opción podado (*pruned node*) permite la ejecución de un nodo completo con la reducción del espacio en disco. El software lleva a cabo la descarga y validación de la cadena de bloques completa, para, una vez alcanzado el umbral de podado, bajar los bloques más antiguos, reteniendo UTXOs y una continuidad de bloques recientes tan amplia como para la operación normal del nodo. El nodo logra mantener las garantías típicas de validar, pero eso sí, con una capacidad reducida para poder devolver el historial

completo. Según Antonopoulos y Harding, este modo es recomendable en ordenadores de baja capacidad de almacenamiento [5].

Txindex

La opción `-txindex` de *Bitcoin Core* permite crear un índice completo de las transacciones y por ello permite realizar búsquedas arbitrarias desde la interfaz RPC[1]. Esta funcionalidad, tal y como dicen Antonopoulos y Harding, responde a la necesidad de contar con herramientas para poder explorar el histórico de operaciones de la red, si bien su uso implica un incremento del almacenamiento requerido y, si se activa a posteriori, la reindexación de la base de datos [5].

Blockfilterindex

La opción `-blockfilterindex` de *Bitcoin Core genera el índice de filtros compactos por bloque* que fueron definidos en la BIP 158, de manera que el nodo puede *ofrecer esos filtros a clientes que hablan el protocolo de la BIP 157*. Dicho de otra forma: el *blockfilterindex* genera un filtro compacto por bloque para que los clientes ligeros (*light clients*) puedan comprobar que una transacción pertenece a un bloque sin necesidad de descargar toda la cadena de bloques. Así se aumenta la eficiencia y la privacidad respecto a la anterior estrategia basada en los *Bloom filters*[2] que definía la BIP 37 [24, 16, 12]. Como indican Antonopoulos y Harding, las BIP constituyen propuestas formales que documentan mejoras o cambios técnicos en el protocolo Bitcoin [5].

UTXO

Una salida no consumida de transacciones (*UTXO* por su acrónimo en inglés) es la cantidad de moneda digital que queda disponible después del transcurso de una transacción. Cada transacción consume salidas no utilizadas anteriormente y produce nuevas salidas que serán posibles utilizar en transacciones posteriores. Se puede asemejar con el cambio en efectivo: al pagar una entidad con un billete de mayor valor, se gasta la entrada por completo, produciendo nuevas salidas gracias al cambio, así como UTXO consumidos y generados en la red Bitcoin [5, 8]. Desde un aspecto operacional, los propios usuarios gestionan los fondos mediante wallets[3] que contienen claves y direcciones, de modo que una dirección es, en la práctica, la versión hasheada de la clave pública que se encuentra asociada a la misma. La protección y la recuperación de las propias claves privadas es

[1]RPC: Se refiere a un protocolo de comunicación que permite a un programa (como una cartera o un nodo) solicitar servicios de manera eficiente a otro programa o servidor a través de una red, normalmente Internet.

[2]Los filtros Bloom son una estructura de datos probabilística que se utiliza para comprobar de manera eficiente si un elemento podría estar en un conjunto de datos sin necesidad de almacenar todos los elementos, por lo cual permite ahorrar espacio y ser muy rápidos; sin embargo, tienen la posibilidad de generar "falsos positivos".

[3]Wallet: billetera digital que permite recibir, enviar y almacenar Bitcoin de forma segura.

fundamental para acceder a los fondos, ya que la pérdida de estas implica la pérdida irreversible de los bitcoins asociados[17].

3.2. Perfiles de usuarios Bitcoin

Tal como se anticipó en el apartado anterior, la definición de los perfiles de usuario hace posible representar los diferentes modos operativos de un nodo Bitcoin. A continuación, se describen los perfiles que en este trabajo se han considerado.

Estándar

El perfil *Estándar* se refiere a un nodo completo de *Bitcoin Core* con la configuración por defecto, ya que dicho perfil sirve de base para poder realizar comparaciones entre los distintos escenarios de prueba, en el cual solo se activó como parámetro adicional `-txindex` (no como personalización, sino como condición necesaria de observabilidad, pues permite registro de datos de indexación y comparación con otros perfiles), y sin poda por defecto. En este sentido, se tiene una copia completa de la *blockchain* y permite una validación conforme a las características del software. Su objetivo está dirigido a usuarios que quieren operar un nodo completo por defecto, pero que permita al mismo tiempo realizar el análisis correspondientes para este TFM, puesto que es un equilibrio entre el hardware utilizado y un comportamiento fiel en la validación. Así lo exponen Antonopoulos y Harding, para quienes los nodos completos son la base de la red: validan cada bloque y transacciones de forma independiente [5]. En este TFM, se denomina *perfil Estándar* al que es utilizado como línea base de comparación para el estudio del rendimiento relativo de los otros perfiles, ya que se caracteriza por un comportamiento habitual de nodo completo a partir de condiciones genéricas.

Validador

El perfil *Validador* se refiere a aquellos que ejecutan nodos completos con el fin de verificar que se hayan seguido correctamente las reglas de consenso. Antonopoulos y Harding resaltan que esta verificación de forma independiente es una tarea necesaria para garantizar la descentralización de Bitcoin, pues no confía en datos de terceros [5]. En esta línea, Satoshi Nakamoto, creador de Bitcoin, y que, según los escritos recopilados por Champagne, enfatiza que la confianza en el sistema proviene precisamente de que cada participante pueda verificar, de manera autónoma, transacciones y bloques [8].

Analista

El perfil *Analista* está dirigido hacia aquellos usuarios que utilizan los datos de la blockchain y que la abordan como método para hacer estudios económicos, sociales o técnicos. Ya se indicaba en los inicios del proyecto que la observación de patrones dentro de las transacciones hacía posible comprender la adopción y qué estaba pasando con la

red [8]. En un sentido complementario, la transparencia en la *blockchain*, convierte sus datos en una fuente de gran valor para la investigación y auditoría [14, 11].

Desarrollador

El perfil *Desarrollador* está destinado a aquellos que crean, mantienen y mejoran el software *Bitcoin Core* o las aplicaciones que derivan de él. Lim y Janse enfatizan que la naturaleza de código abierto del ecosistema blockchain ha provocado mejoras técnicas, pero también debates sobre la gobernanza y la descentralización [14]. En el mismo sentido, Antonopoulos y Harding describen que la evolución de Bitcoin se desarrolla mediante BIP discutidos y revisados por toda la comunidad [5], y en el sentido de lo que exponen los escritos de Satoshi que recogen Champagne, ya se hacía hincapié en la colaboración a través de los desarrolladores [8].

Lightning

El perfil *Lightning* se encuentra dirigido a usuarios que quieren conseguir un mayor volumen de transacciones a gran velocidad y por comisiones muy bajas mediante un sistema de canales de pago, fuera de la cadena principal. Estas soluciones se encuentran en línea con los mecanismos de segunda capa, ideados para resolver los problemas de escalabilidad de bitcoin. En este sentido, Drescher explica el método de escalamiento por capas [10], y Lim y Janse reconocen a la *Lightning Network* como método cuya implementación destacada [14]. Finalmente, caracterizaciones empíricas de la red Lightning (gossip[4], parámetros de canales, distribuciones de implementaciones y actividad) pueden encontrarse en [25].

Educador

Este perfil *Educador* está dirigido para quienes usan la tecnología como recurso didáctico. Bartolomé y Moral argumentan que la blockchain puede generar contextos educativos innovadores que generen experiencias de aprendizaje distintas y favorezcan el mejor conocimiento del funcionamiento de tecnologías disruptivas [6].

3.3. Métricas de rendimiento

Uso de CPU

La verificación de bloques y transacciones conlleva una carga considerable de la CPU, en particular en la fase de sincronización inicial de un nodo. Antonopoulos y Harding recuerdan que la negociación criptográfica está asociada a las operaciones de *hashing* de

[4]El protocolo "Gossip"se refiere a un método de comunicación distribuida inspirado en la forma en que se propagan los rumores. En este método, los nodos de la red Bitcoin comparten información sobre transacciones y bloques. De esta manera, todos los participantes obtienen una vista consistente y actualizada de la blockchain.

todos los bloques y transacciones [5]. En paralelo, Tschorsch y Scheuermann hacen notar cómo la prueba de trabajo acoplada a la función hash criptográfica SHA-256[5] conlleva un coste computacional deliberado, el cual se regulará periódicamente por el sistema mediante el mecanismo de ajuste de dificultad del PoW[6][21]. De esta manera, la carga computacional de un nodo variará directamente con la velocidad de validación y propagación de nuevos bloques.

Uso de RAM

La utilización de memoria se encuentra determinada fundamentalmente por los siguientes dos motivos: (i) la *caché* de la base de datos (`dbcache`) usada para validar el conjunto UTXO y (ii) el *mempool* que se mantiene en la RAM. El conjunto UTXO queda almacenado de forma persistente en disco y el acceso se acelera mediante cachés configurables. En el caso del *mempool*, este reside en memoria, por lo tanto, la RAM disponible influye en la propagación y en la selección de transacciones [5, 21, 14]. Por lo que el tener una configuración adecuada de memoria disminuye las lecturas de disco y aumenta el rendimiento de validación.

Uso de disco

El almacenamiento local de la blockchain representa uno de los mayores retos en términos de hardware. Ya en el año 2014, Champagne alertaba que la base de datos de Bitcoin aumentaba de manera ininterrumpida, lo que dificulta que los usuarios en general puedan operar nodos completos [8] y, a este aspecto, Tschorsch y Scheuermann mencionan también que el tamaño de la cadena en el año 2016 ya alcanzaba decenas de gigabytes, y sigue creciendo sin parar, dando lugar a problemas de escalabilidad y de capacidad de almacenamiento a largo plazo [21]. Además, desde la visión de la gobernanza, Campbell-Verduyn subraya que esta alta exigencia técnica tiende a restringir la operación de nodos a actores que se encuentran en mejores condiciones de infraestructura, generando efectos no deseados en la descentralización de la red [7].

Uso de red

Los nodos que forman la red de Bitcoin van intercambiando bloques y transacciones a través de una red *peer-to-peer*. La sincronización de la red global es de hecho un tráfico relevante o importante para la misma, hasta el punto de que Antonopoulos y Harding hacen hincapié en que el propio rendimiento de la red dependía de la calidad y ancho de banda disponible [5]. Tschorsch y Scheuermann describen además que los retardos

[5]SHA-256 es el algoritmo de hash criptográfico que Bitcoin utiliza para la seguridad, generando una firma digital única de 256 bits para los datos de las transacciones, proceso que es conocido como minería y es crucial para poder verificar la autenticidad de las transacciones, asegurar la integridad de la cadena de bloques y prevenir el doble gasto.

[6]Proof of Work: es un mecanismo de consenso en blockchain que utiliza el poder de procesamiento informático para validar y agregar nuevas transacciones a la cadena de bloques.

de propagación en la red P2P pueden generar bifurcamientos, afectando la latencia de validación y, la robustez del consenso distribuido [21].

Tiempo de sincronización

El tiempo que le llevará a un nodo que descargue y verifique la totalidad de la cadena de bloques está en función de la capacidad de la CPU, de la RAM, del disco y de la red. Drescher también expone que la sincronización inicial puede suponer una dificultad para la entrada de nuevos participantes [10]. Del modo más análogo, los autores Tschorsch y Scheuermann exponen en qué medida los retardos en la propagación de bloques y el esfuerzo afectan a la usabilidad de la sincronización, mientras que Campbell-Verduyn muestra el impacto de dichas dificultades en la accesibilidad y en la distribución global de los nodos [7].

3.4. Automatización en Linux

Scripts Bash

El uso de *scripts* en Bash ofrece la posibilidad de automatizar tareas que pueden ser repetitivas y críticas, como pueden ser el arranque del nodo, hacer copias de seguridad, capturar métricas del rendimiento, etc. La automatización de las tareas evita la posibilidad de incurrir en errores humanos y mejora la operativa [14]. En la administración del sistema Linux, el scripting en Bash se usaba para encadenar órdenes, planificar rutinas, etc. [2].

Systemd

El sistema de inicio `systemd` en Linux facilita gestionar *Bitcoin Core* como un servicio, permitiendo su gestión de arranque al iniciar el sistema, una vigilancia permanente, reinicio bajo fallos, gestión de dependencias entre depósitos de servicios, etc. Es el sistema de inicio por defecto en las principales distribuciones Linux y hace de PID 1, gestionando los servicios mediante archivos de unidades (`*.service`), controla recursos mediante archivos necesarios y ofrece herramientas como `systemctl`, `journalctl` y `systemd-analyze` [20]. Para la integración operativa del sistema, el referente son las guías de administración de Linux (p. ej., creación de `unit files` con `ExecStart`, `Restart`, `WantedBy`) [2].

Monitorización

La monitorización es el proceso que consiste en recoger y estudiar métricas en tiempo real (CPU, RAM, disco, red) con el objetivo de determinar el rendimiento de un nodo y también para prevenir incidencias. La observación de datos en tiempo real, en educación e investigación aplicada, facilita la comprensión de tecnologías complejas [6]. En la gestión diaria de sistemas Linux, herramientas como `top`, `htop`, `vmstat` o `df` constituyen un elemento esencial para detectar cuellos de botella y decidir la correcta decisión operativa [2].

4: Técnicas y herramientas

4.1. Lenguajes y herramientas

El núcleo de este trabajo fue desarrollado mediante un **script en Bash** llamado `script_tfm.sh`, el cual incorpora todo el proceso de instalación y configuración de Bitcoin Core en sistemas Unix. Incorpora las funciones para la verificación de los requerimientos, la instalación de las dependencias, la compilación desde código fuente, la generación de los archivos de configuración, la elaboración de los perfiles de usuario y el registro de las métricas de rendimiento en un archivo en formato CSV. En administración de sistemas Linux, los *scripts* en Bash son un mecanismo estándar para conseguir la reproducibilidad y la transparencia al administrar tareas y flujos [2]; complementariamente, en los proyectos blockchain se prioriza la simplicidad operativa para de este modo facilitar la transferencia de conocimiento [14].

Compilar **Bitcoin Core v29.0** desde su código fuente posibilitó tener un control exhaustivo de las características, sin depender de binarios precompilados, así como la posibilidad de habilitar características como `-txindex` o `-blockfilterindex`. Esta práctica se alinea con el énfasis del ecosistema Bitcoin en el *software* auditable y verificable por la comunidad [8], así como en la documentación de versiones recientes [9].

Para la gestión de la puesta en marcha del nodo, fue adoptando **systemd** mediante una unidad de servicio orientada a `bitcoind`. Esta adopción permite el arranque automático del sistema, su detención y reinicio, así como la supervisión del proceso. El propio `systemd` es el proceso de inicialización por defecto de las principales distribuciones, se ejecuta como PID 1, organiza los servicios a partir de los ficheros de unidad a su disposición, gestiona dependencias y proporciona utilidades como`systemctl`, `journalctl` y `systemd-analyze` [20]. En su utilización es coincidente con las buenas prácticas de administración de Linux [2].

La **monitorización del sistema** se integró en el script `script_tfm.sh`, mediante la invocación de utilidades estándar de Linux (`top`, `free`, `df`, `uptime`, así como tras consultas a `/proc` y `/sys`), que permiten observar uso de CPU, memoria, disco y tráfico de red. El

Herramientas	**Automatización**	**Monitorización**	**Análisis**	**Servicio**
Bash (`script_tfm.sh`)	✓	✓		
Bitcoin Core (compilado desde fuente, v29.0)				✓
`systemd` (unidad `bitcoind.service`)				✓
Utilidades Linux (`top`, `free`, `df`, `uptime`, `jq`)		✓		
Formato CSV (salida de métricas)		✓	✓	
Python (`graficos.py`, pandas, matplotlib)			✓	

Tabla 4.1: Herramientas y tecnologías utilizadas en el TFM

seguimiento de métricas cuantitativas es útil para poder entender el funcionamiento de sistemas distribuidos y identificar los cuellos de botella de forma anticipada [10, 2].

Los resultados de los datos obtenidos de la monitorización se guardaron en ficheros **CSV**, que son un formato sencillo e interoperable con herramientas de análisis de datos. En el marco de docencia e investigación aplicada, elegir un formato accesible puede contribuir a la comprensión y a la reutilización de los mismos [6].

De los CSV se utilizó un **script en Python** (`graficos.py`) para la representación visual. Se procesaron los datos con `pandas` y se generaron representaciones temporales y comparativas según el perfil de los usuarios con `matplotlib`. Las herramientas de análisis permiten transformar métricas de sistemas descentralizados en conocimiento aplicable [10] y se alinean con la filosofía de apertura y verificabilidad del ecosistema [5].

En la Tabla 4.1 se presenta un resumen de las herramientas y tecnologías utilizadas, haciendo las distinciones según su función principal.

4.2. Entorno de pruebas

Se llevaron a cabo las pruebas en un entorno controlado, con suficiente *hardware* y *software* adecuado para realizar y examinar un nodo completo sin limitaciones. Se utilizó **Ubuntu 24.04.2 LTS** con el kernel `6.8.0-63-generic`; un **Intel Core i5-8400** (6 núcleos, 2.80 GHz) y **16 GB de RAM** (15 GB disponibles en pruebas). En almacenamiento, un **disco sólido de 1,8 TB** con 1,7 TB libres al inicio, lo cual fue suficiente para la cadena de bloques y archivos auxiliares generados. Como *software* de referencia se utilizó **Bitcoin Core v29.0** compilado desde fuente para garantizar realizar un control total de las opciones de configuración.

Recurso	Detalle
Sistema operativo	Ubuntu 24.04.2 LTS, kernel 6.8.0-63 (arquitectura x86_64)
CPU	Intel Core i5-8400, 6 núcleos a 2.80 GHz
Memoria RAM	16 GB totales (15 GB utilizables)
Disco	1,8 TB total; 1,7 TB libre al inicio; partición /dev/sda2; sistema de archivos ext4
Software	Bitcoin Core v29.0, compilado desde fuente

Tabla 4.2: Entorno de pruebas empleado

Las especificaciones del entorno son las que se recojan en la Tabla 4.2.

4.3. Alternativas consideradas

Virtualización y contenedores

Se barajó como opción la encapsulación del nodo con contenedores (Docker, por ejemplo) para su despliegue, pero terminó optando por ejecución *nativa* con objeto de no introducir capas de abstracción extra en la medición y así determinar mejor el impacto de CPU, la memoria, el disco y la red en el tiempo de sincronización, dado que, además, esta decisión metodológica favorece la posibilidad de comparación en mediciones de rendimiento en entornos controlados[3].

Lenguajes de programación

Se barajó como opción `C` como lenguaje de programación para crear herramientas de automatización y monitorización; finalmente, se optó por Bash y Python por su sencillez y su integración en el ecosistema Linux, lo que permite replicabilidad y transferencia de conocimiento fácilmente [14, 2].

Sistemas de monitorización externos

Así mismo, se contemplaron opciones ampliamente utilizadas, como lo son ELK[1], Zabbix[2], entre otras, pero finalmente se optó por una implementación liviana basada en CSV junto con Python, tanto por la facilidad de instalación como por la rapidez de

[1]ELK: Acrónimo de la pila Elasticsearch, Logstash y Kibana, que son un conjunto de herramientas de código abierto para recopilar, analizar y visualizar datos, especialmente registros (logs).

[2]Zabbix: Software de monitoreo de código abierto de nivel empresarial que sirve para supervisar redes, servidores, aplicaciones y otros dispositivos en tiempo real.

obtención de resultados, sobre todo en el contexto de la docencia y en la investigación aplicada, en donde interesa poner el foco en la comprensión de la tecnología más que en la complejidad de la infraestructura [6].

4.4. Síntesis

Los temas tratados en este capítulo se refieren a las decisiones técnicas que dan soporte al proyecto: una línea de trabajo austera y auditable mediante Bash, para la automatización de la instalación, de la configuración y la captura de métricas; Python (pandas/matplotlib) para el análisis y la visualización; y `systemd` para el funcionamiento estable del servicio `bitcoind`. La elección de CSV como formato de datos persigue la neutralidad y la interoperabilidad, facilitando el hecho de poder ser inspeccionado manualmente y ser reutilizado en diferentes entornos académicos. En conjunto, la arquitectura persigue la sencillez, la capacidad de reproducir lo documentado y el control del software (compilado a partir de la fuente y habilitando las opciones predeterminadas).

Para no introducir capas que sesgaran las mediciones, se optó por ejecución nativa (sin contenedores) y herramientas estándar de Linux para obtención de monitorización ligera (CPU, RAM, disco, red); se desechaban de esta manera otras alternativas más complejas que no aportaban valor diferencial al propósito del trabajo. De esta forma, se busca justificar una base técnica adecuada, portable y defendible, coherente con los propósitos del TFM y apta para poder comparar perfiles de uso en condiciones controladas.

5: Aspectos relevantes del desarrollo del proyecto

Este capítulo trata los elementos más significativos que se contemplan en el marco del Trabajo Fin de Máster, pero adoptando un enfoque más práctico y reflexivo, considerando las decisiones que fueron tomadas dentro del proyecto y las dificultades que aparecieron, y por supuesto las soluciones que fueron llevándose a cabo.

5.1. Ciclo de vida y enfoque metodológico

El desarrollo del proyecto se llevó a cabo mediante un **enfoque iterativo e incremental**, que daba por hecho un ciclo de vida flexible, en tanto que cada iteración iría aportando nuevas funcionalidades y solucionando problemas de iteraciones anteriores.

De la estancia de I+D+i al TFM

La fase inicial del proyecto fue la estancia de **I+D+i**, en la cual se documentó la **instalación y configuración manual de un nodo Bitcoin Core en Linux**, fue una fase de carácter formativo, la cual sirvió para comprender la lógica interna del propio software, los ficheros que se van generando durante la sincronización y las opciones básicas de configuración del nodo. El resultado de la fase de trabajo fue un informe técnico pensado como una guía práctica.

El Trabajo Fin de Máster se enfocó en dar un paso más: convertir la experiencia en una **solución automatizada y evaluable**. Ya no se trataba de instalar solo un nodo; se tenía como fin el crear un mecanismo de despliegue para diferentes perfiles de usuario y, al mismo tiempo, medir de manera sistemática el impacto de las configuraciones sobre el consumo de recursos, todo ello siguiendo los consejos dados por el profesor guía del proyecto.

Iteraciones del desarrollo

El proceso se organizó en varias iteraciones consecutivas:

1. **Instalación manual y primeros ensayos a partir de la estancia de I+D+i.** Reproducir la instalación de Bitcoin Core a partir del código fuente, además de validar la conectividad de Bitcoin Core y su funcionamiento básico.
2. **Automatización mediante `script_tfm.sh`.** Primera versión de script en Bash con el cual poder ejecutar la instalación, la configuración y el despliegue como servicio de `systemd`.
3. **Definición de diferentes perfiles de usuario.** Consideración de diferentes tipos de configuraciones de caso real (Validador, Analista, dApps, Lightning, Educador y Estándar), de acuerdo con las indicaciones del tutor del proyecto.
4. **Integración de la monitorización.** Evolución a partir de un script externo que se ejecuta en la terminal y que finalmente termina siendo una solución integrada dentro del `script_tfm.sh`, que se ejecuta junto a un flag opcional `-log`.
5. **Registro y análisis de métricas.** Generación de ficheros CSV delimitados por `";"` a partir del flag opcional -log, para de de este modo evitar errores en el parseo desde Python, y análisis posterior a través de `graficos.py`, para poder generar los gráficos de cara a un análisis posterior entre los distintos perfiles.
6. **Comparación entre perfiles.** Obtención de gráficos comparativos y tablas de resumen en base a los ficheros que se han agregado con la finalidad de poder realizar una comparación real de consumos entre las distintas configuraciones.

Justificación del enfoque de ciclo de vida

Los motivos que justifican el ciclo de vida implementado son los siguientes:

- **Naturaleza experimental:** La monitorización de un nodo Bitcoin bajo parámetros muy poco documentados en un contexto real (para la discusión de la comparabilidad de métricas, ver [11]).
- **Necesidad de depuración incremental:** El script fue elaborado mediante prueba y error, corrigiendo uno a uno los problemas de sintaxis, problemas de permisos, problemas con el formato de los datos, etc.
- **Flexibilidad:** Ha permitido incorporar mejoras que inicialmente no estaban previstas, como, por ejemplo, el flag `-log` y la separación clara entre las fases de la *sync* y *post-sync*.
- **Reproducibilidad:** La combinación de automatización y registro estándar permite a otros volver a realizar los experimentos bajo condiciones similares.

Representación visual del ciclo

Para ilustrar este proceso, la Figura 5.1 muestra un esquema simplificado del ciclo de vida aplicado en el proyecto.

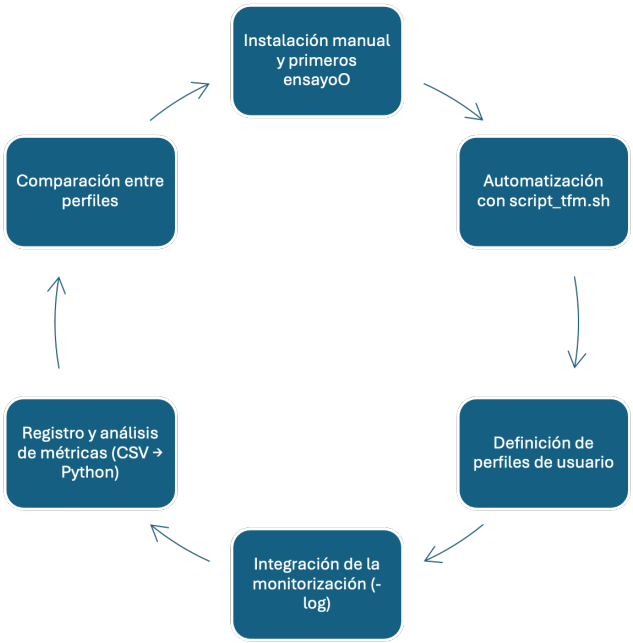

Figura 5.1: Flujo iterativo del ciclo de vida seguido en el proyecto.

Síntesis

Para finalizar, el progreso del proyecto no siguió un modelo rígido, sino que fue el resultado de un proceso de **iteraciones continuas y aprendizaje práctico**. Cada una de las fases fue mejorada tras analizar la anterior, hasta alcanzar una solución completa, donde se combinaron automatizaciones, monitorizaciones y análisis comparativo, de modo de cumplir con los objetivos académicos buscados para el Trabajo Fin de Máster.

5.2. Análisis y diseño de la solución

La solución pensada se sustenta en la necesidad de poder **automatizar el despliegue de nodos Bitcoin** para distintos perfiles de usuario y, al mismo tiempo, medir el consumo de los recursos de forma sistemática para cada perfil. Para ello, se definió una arquitectura modular en la que cada componente implementado realiza una función muy específica.

Arquitectura general

La arquitectura del sistema se configuró en función de los siguientes elementos:

- **Bitcoin Core (v29.0 compilado desde fuente):** Software base que implementa el nodo completo y proporciona la interfaz RPC; la compilación desde la fuente asegura la transparencia, el control sobre opciones de configuración y la reproducibilidad [9].
- **Script Bash (`script_tfm.sh`):** Núcleo de la automatización, que se encarga de la instalación de dependencias, de la compilación, de la configuración de perfiles, del lanzamiento mediante `systemd` y de la captura de métricas en ficheros CSV.
- **Gestor de servicios (`systemd`):** encargado de lanzar `bitcoind` como servicio persistente, de llevar a cabo el reinicio automático y de realizar control mediante a supervisión estándar [20].
- **Monitorización ligera en CSV:** Registro periódico de métricas de CPU, RAM, disco, red, procesos y el progreso de sincronización (`verificationprogress`); los ficheros se generan con separador `";"` para asegurar que en Python puedan ser leídos correctamente[1].
- **Script en Python (`graficos.py`):** Se encarga del análisis de los CSV; realizando estadísticas descriptivas y generando las gráficas de series temporales para realizar las comparativas entre perfiles.

La Figura 5.2 muestra de forma esquemática la arquitectura de la solución y la interacción entre sus componentes.

[1]El uso de `";"` como delimitador evita el conflicto con las comas decimales empleadas en el formato regional español.

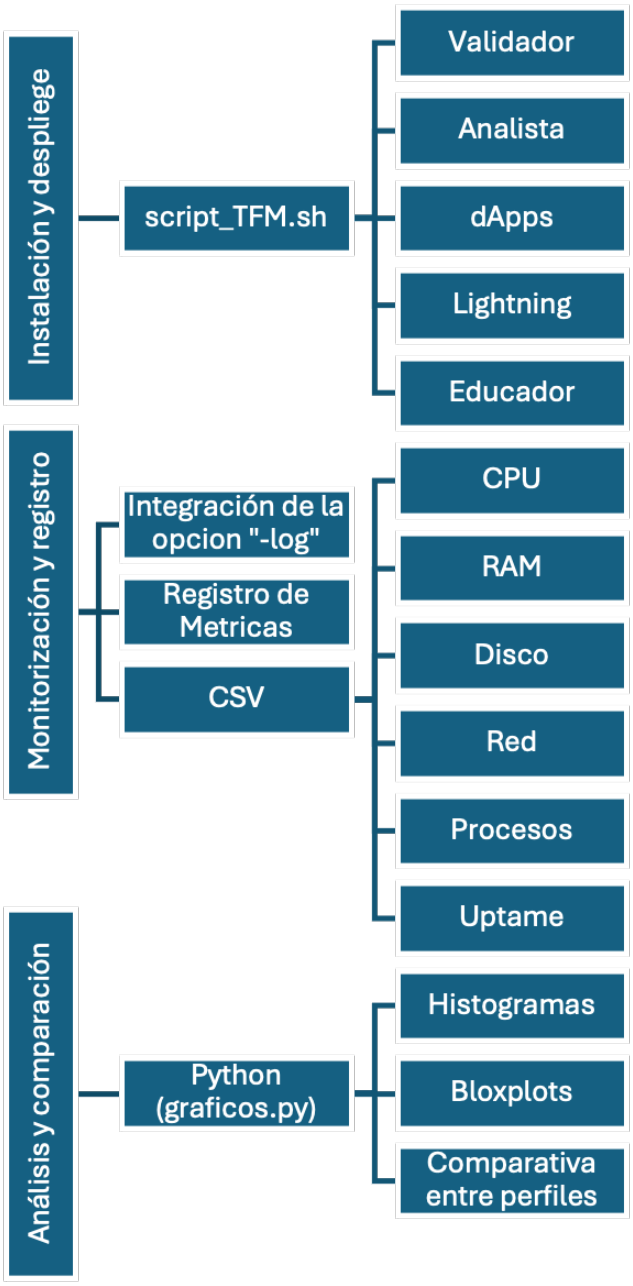

Figura 5.2: Arquitectura modular de la solución desarrollada.

Definición de perfiles de usuario

Uno de los aspectos más determinantes del diseño propuesto fue la inclusión de los **perfiles de usuario**, que proporcionaban la posibilidad de adaptar de forma automática los parámetros para `bitcoin.conf` en función de los requisitos que nuestra situación requiere, tal como fue recomendado por el tutor. La Tabla 5.1 recoge los distintos perfiles que se han considerado, así como los principales parámetros técnicos y el caso de uso que

Perfil	**Parámetros técnicos (`bitcoin.conf`)**	**Caso de uso típico**
Validador	Para La validación más fuerte, se establece `prune=0`, validando bloques y scripts completos sin índices suplementarios.	Usuarios avanzados y grupos que deseen la máxima soberanía y seguridad verificando toda la cadena de bloques.
Analista	Se configura `prune=0`, `txindex=1`, `blockfilterindex=1` [16, 24], posibilidad de enlazar una base de datos externa.	Investigación académica y empresas de análisis de los blockchains que desean acceso último a toda la historia de las transacciones.
Desarrollador (dApps)	Nodo podado que brinda acceso completo a la interfaz RPC, posibilidades de consultas rapidas (`getrawtransaction`, `estimatesmartfee`).	Los programadores que desarrollan billeteras, integración de pagos, o aplicaciones descentralizadas.
Lightning	Pruning moderado, sincronización rápida, acceso al mempool	Los servidores de nodos Lightning Network que desean un backend robusto que mantenga abiertos y refrescados los canales Lightning.
Educador	Nodo podado, validación ligera, posibilidad de suprimir scripts para así permitir la sincronización rápida.	Los educadores y alumnos que requieren un nodo ligero dentro de la aplicación de pequeñas demostraciones o de técnicas de formación en la práctica.
Estándar	Configuración equilibrada (con los parámetros por defecto pero con `txindex=1` de modo que permita un análisis homogéneo y consultas sobre el histórico completo sin reindexación posterior).	Usuarios comunes que quieren un nodo completo de referencia, pero sin requerimientos técnicos avanzados. configuraiones espelizadas.

Tabla 5.1: Perfiles de usuario y parámetros de configuración

nos servía de guía para cada uno de ellos (para fundamentos de nodo completo y operación de red, véase [5]).

Justificación del diseño

El diseño elegido presenta varias ventajas:

- **Modularidad:** La existencia de componentes individualizados (instalación, servicio, monitorización, análisis) permite que se puedan desacoplar fácilmente unos de otros para dar mayor flexibilidad al mantenimiento y a las ampliaciones futuras.
- **Flexibilidad:** Los perfiles permiten, sin tener que realizar cambios manualmente en la configuración, cubrir diferentes escenarios.
- **Reproducibilidad:** La combinación de Bash y Python asegura que la combinación de los scripts permita replicar los resultados en condiciones similares por cualquier investigador.
- **Orientación práctica:** La solución diseñada no solo queda en la teoría, sino que ofrece una herramienta aplicable en la docencia, investigación o en despliegues productivos.

5.3. Implementación

La etapa de implementación se basó en la conversión del diseño conceptual en un conjunto de medios que se ocupasen de que la instalación y configuración de los nodos Bitcoin Core, la monitorización y análisis de los mismos tuviese lugar de forma automatizada. Para ello se desarrollaron dos partes de software propias: el **script en Bash** `script_tfm.sh` y el **script en Python** `graficos.py`.

Estructura del `script_tfm.sh`

El script está escrito en lenguaje Bash, y es el corazón mismo del proceso de automatización buscado en el TFM, el cual está sistematizado modularmente mediante las funciones que encapsulan las distintas tareas, contribuyendo a un mejor ajuste de las distintas tareas, permitiendo la lectura, el mantenimiento y las posibilidades de mejora futura.

La Tabla 5.2 resume las funciones principales y su propósito dentro del flujo general de ejecución.

Evolución del script y depuración

La elaboración del `script_tfm.sh` fue llevada a cabo gracias a un proceso de **prueba y error**, ya que cada una de las ejecuciones de este sistema daba lugar a barreras de sintaxis o de lógica que se corregían hasta dar con un eficiente funcionamiento.

Dos aspectos resultaron ser especialmente importantes:

- **Separador de CSV:** En la primera versión, el uso de la coma como separador producía errores de parsing en Python, pues entendía, por ejemplo, los float como nuevas columnas. Por ello, se optó por escoger el carácter separador **";"**, pues esta solución garantizaba poder leerlo correctamente con `pandas`.

Función	**Propósito**
`check_requirements()`	Esta función se ejecuta siempre al inicio, verificando la versión de Linux, la arquitectura, espacio en disco, la RAM total, las librerías necesarias y compiladores que seran usados, para indicar si es o no posible instalar bitcoin-core en el equipo.
`install_dependencies`	Instala los paquetes requeridos para poder compilar el software bitcoin-core, esto incluye algunas librerías, herramientas de compilación y las utilidades de red para el correcto funcionamiento del script en general.
`compile_bitcoin()`	Descarga y compila Bitcoin Core 29.0 desde la fuente oficial en github.
`configure_profile()`	Esta funcíón genera el fichero bitcoin.conf con los parámetros de funcionamiento como `prune`, `txindex` o `blockfilterindex` correspondientes al perfil deseado, para que una vez iniciada la ejecución de bitcoin-core, ocupe los datos del perfil seleccionado.
`setup_systemd()`	Esta función crea el fichero `bitcoind.service`, y habilita este servicio en el arranque, permitiendo su control a través de la herramienta systemctl, con ello es mucho mas facil manejar bitcoin-core.
`start_monitoring()`	Esta función inicializa la captura de métricas de CPU, RAM, disco, red, procesos y verificationprogress, esto ocurre una vez iniciada la ejecucion de bitcoin-core, siempre y cuando en la ejecucion de `script_tfm.sh` se agrege el parametro `-log`, de modo que toda la información quede registrada en un archivo CSV.
`main()`	Esta es la función principal, que contiene la lógica general de la herramienta: procesamiento de argumentos de entrada, selección del perfil, ejecución de instalación, configuración del nodo, habilitación del servicio y, finalmente, si corresponde, la inicialización de la monitorización.

Tabla 5.2: Funciones principales del `script_tfm.sh`

- **Integración de la monitorización:** En los primeros pasos, se probaba un script que se lanzaba en otra terminal para poder registrar métricas, pero para mejorar el proceso se optó por implementar la monitorización directamente en el propio `script_tfm.sh`, que podía habilitarse haciendo uso del flag opcional `-log`, pues al ejecutar `"./script_tfm.sh -log"`, permitía registrar las métricas deseadas en un CSV[2], y que termina su primera ejecución una vez ya terminada la correcta sincronización del nodo bitcoin, y su segunda ejecución 1 hora posterior al término de la correcta sincronización.

Monitorización integrada

El proceso de monitorización se lleva a cabo empleando las siguientes utilidades estándar en Linux: `top`, `free`, `df`, `uptime`; y lecturas de `/proc` y `/sys`; que, en último término, permiten la captura periódica de lo siguiente [2]:

- **CPU:** Porcentaje de uso global y por núcleo.
- **RAM:** Memoria total, utilizada y disponible.
- **Disco:** Espacio ocupado por la cadena de bloques.
- **Red:** Tráfico de red entrante y saliente (en kilobytes).
- **Procesos:** Número total de procesos activos.
- **Progreso:** Indicador `verificationprogress` de Bitcoin Core.

Los datos adquiridos se almacenan en CSV, muestreando cada 60 segundos, diferenciando dos fases: la **sincronización inicial** (*sync*) y el estado **post-sincronización** (*post-sync*), que corresponde al análisis de 60 minutos tras terminar la IBD (descarga inicial de bloques).

Procesamiento con `graficos.py`

El resultado del análisis se concretó en el script `graficos.py`, el cual fue creado en Python a partir de `pandas` y `matplotlib`; se incluyen las principales funcionalidades gráficas:

- Limpieza y normalización de los CSV generados.
- Cálculo de estadísticas descriptivas: media, mediana, desviación típica, valores mínimos y máximos.

[2]El registro se implementó con comandos estándar de GNU/Linux, evitando librerías externas o demonios adicionales para garantizar entornos reproducibles.

- Gráficas: Series temporales y gráficas comparativas entre los distintos perfiles.
- Exportación de tablas resumen (ej. `tabla_comparativa.csv`), que permiten sintetizar las métricas medias por perfil y fase para poder realizar comparaciones rápidas a simple vista.

Síntesis

La implementación se caracterizó por su **modularidad**, interacción junto a la **integración de la monitorización ligera** y la **adaptación a perfiles de usuario**; el `script_tfm.sh` mantiene la reproducibilidad del despliegue de la información y de la ejecución de la obtención de métricas. Por su parte, `graficos.py` se encarga de convertir la información de los datos en bruto a gráficos ajustados para el análisis del proyecto.

5.4. Resultados experimentales

El análisis de los resultados obtenidos se presenta en dos etapas: primero se estudia la **sincronización inicial** (*Initial Block Download, IBD*), que es el proceso más intensivo en recursos, y posteriormente se analiza la fase de **post-sincronización**, donde el nodo opera en régimen estable. En ambos casos se presentan los resultados del perfil estándar como línea base, seguido de una comparativa entre perfiles.

Proceso de sincronización inicial

Uso de CPU durante la sincronización

La comparativa global (Fig. 5.3) evidencia claramente que el perfil **Educador** es el más intensivo en el uso de la CPU, llegando a alcanzar hasta el 54 %, mientras que **dApps** y **Lightning** tienen los porcentajes más bajos, de un 37 %. El perfil **Estándar** es el que ocupa una posición intermedia de un 38 %, junto a **Analista** y **Validador**. Los gráficos individuales permiten observar de manera más detallada la carga de cada perfil: el **Estándar** (Fig. 5.4) mantiene un patrón estable; el **Educador** (Fig. 5.5) presenta picos frecuentes superiores al 80 %, lo que da cuenta claramente de su exigencia; el perfil de **dApps** (Fig. 5.6) presenta curvas más controladas, sin grandes picos de requerimiento.

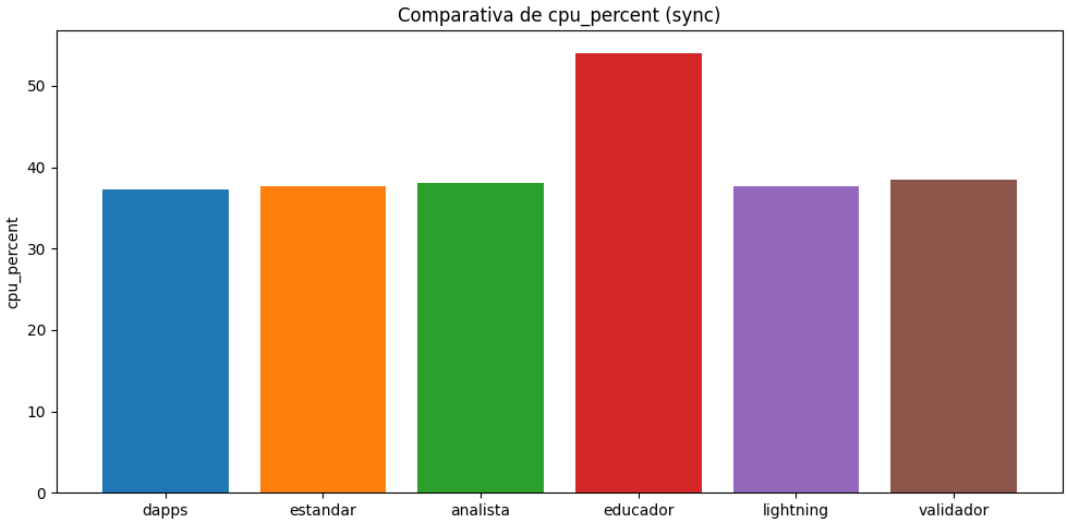

Figura 5.3: Consumo medio de CPU durante la sincronización por perfil.

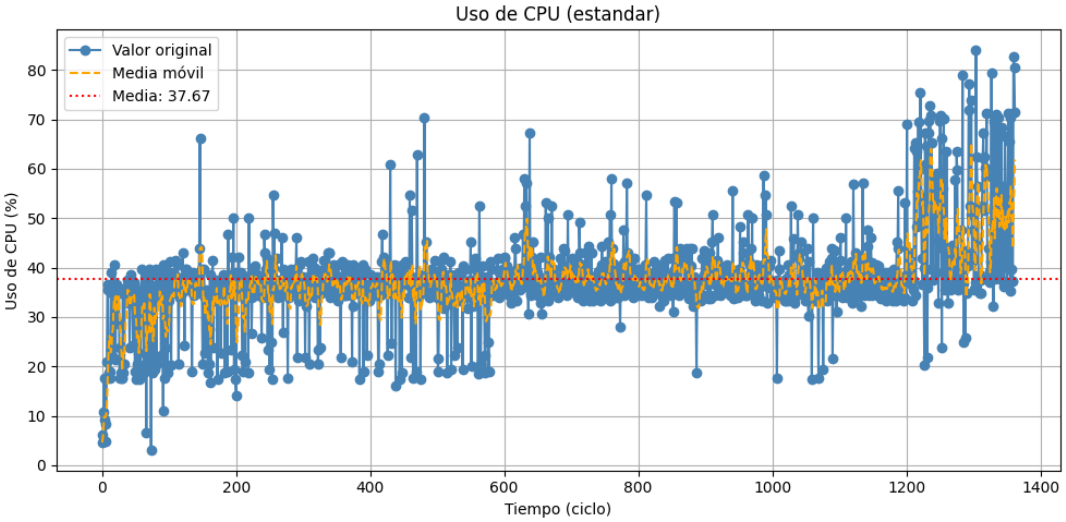

Figura 5.4: Uso de CPU durante la sincronización inicial (perfil estándar).

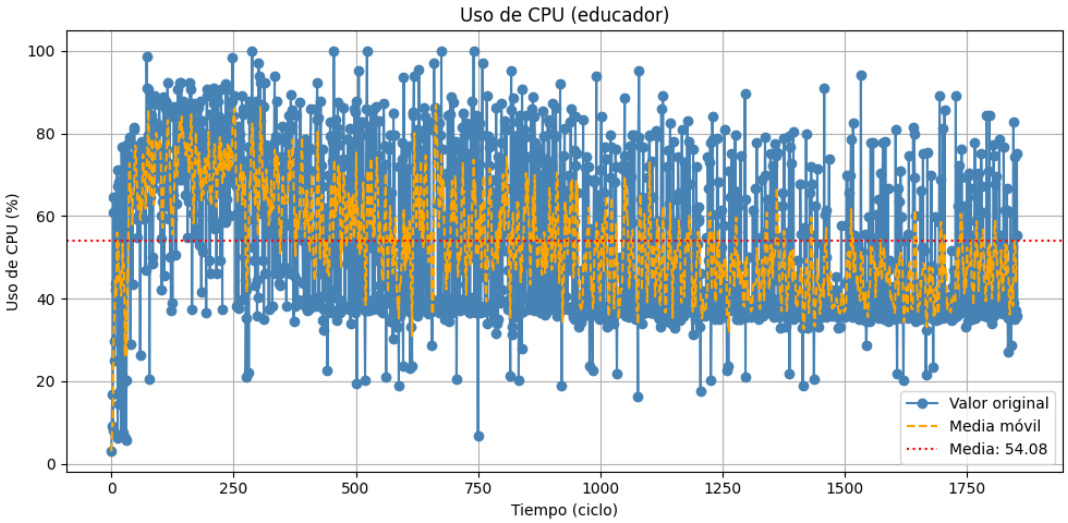

Figura 5.5: Uso de CPU durante la sincronización inicial (perfil educador, el más alto).

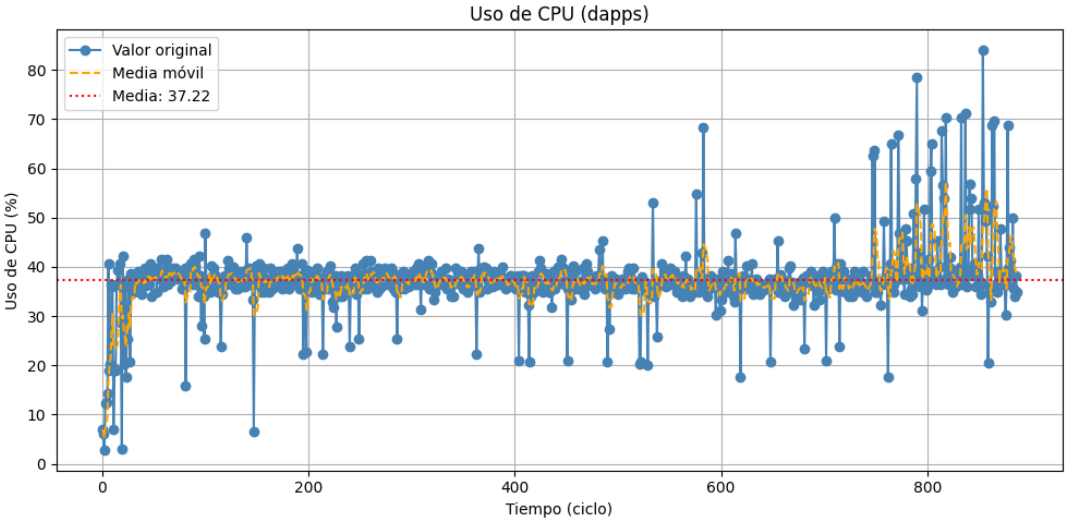

Figura 5.6: Uso de CPU durante la sincronización inicial (perfil dApps, el más bajo).

Uso de RAM durante la sincronización

La comparativa general (Fig. 5.7) muestra que la mayor parte de los perfiles tienen un consumo de RAM muy parecido, con diferencias solo de un par de puntos porcentuales. El **perfil Estándar** se encuentra en torno al 13 % (Fig. 5.8) y, por tanto, se sitúa como una referencia estable. El **Educador** (Fig. 5.9) tiene el valor medio más bajo, de 10.8 %, que da cuenta de su idoneidad para los entornos de memoria restringida. En el extremo opuesto, el **Lightning** (Fig. 5.10) tiene el valor medio más alto (12.9 %), consecuencia de la gestión extra de canales de pago. Por su parte, los perfiles **Analista**, **Validador** y

dApps se encuentran en una posición intermedia, muy cerca del estándar, lo que quiere decir que el consumo de RAM no es un factor crítico de diferenciación en dicha fase de sincronización, a diferencia de CPU o disco. Resumiendo, el perfil estándar mantiene un buen valor medio en lo que respecta a la memoria, el **Educador** sobresale muy bien por ser ligero y **Lightning** se sitúa en el borde superior de este recurso.

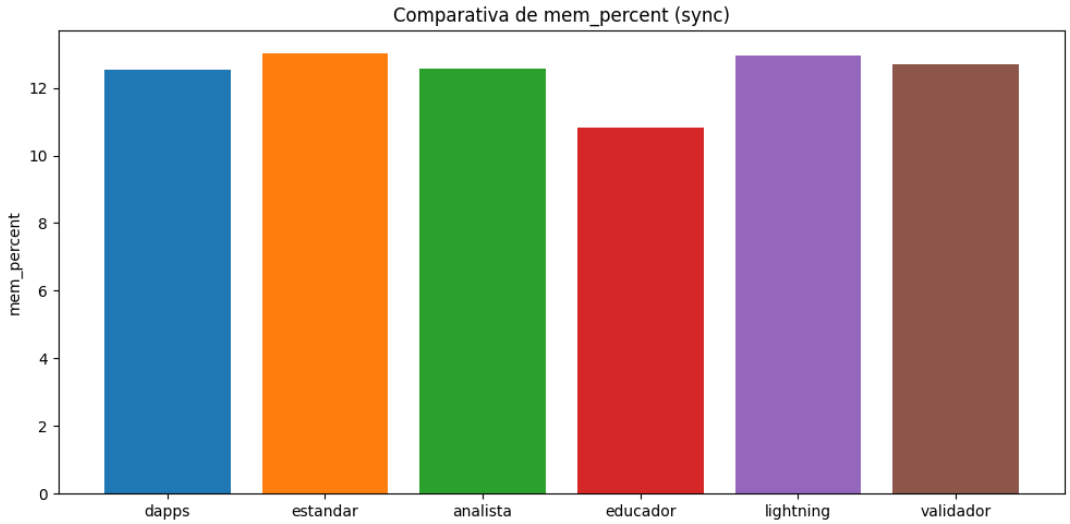

Figura 5.7: Consumo medio de RAM durante la sincronización por perfil.

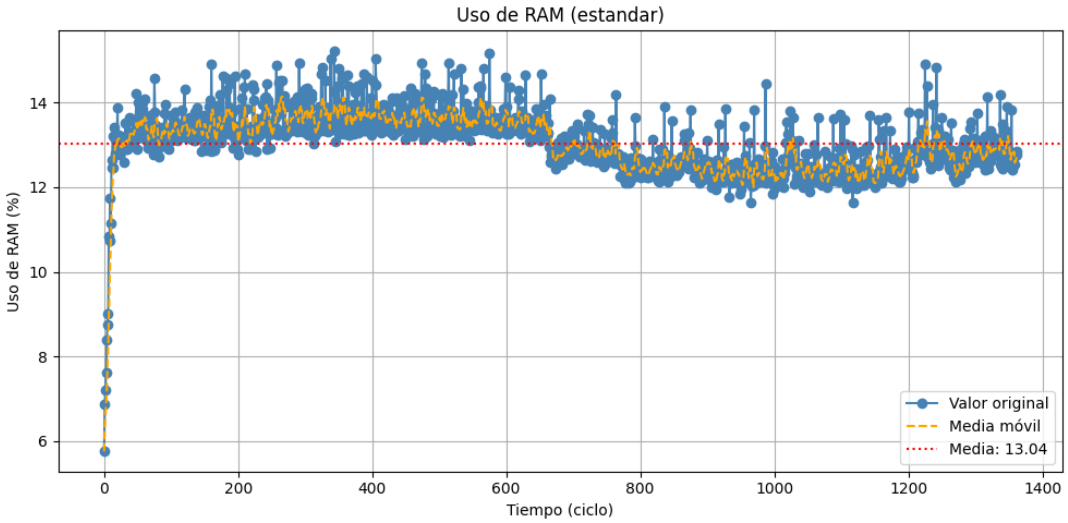

Figura 5.8: Uso de RAM durante la sincronización inicial (perfil estándar).

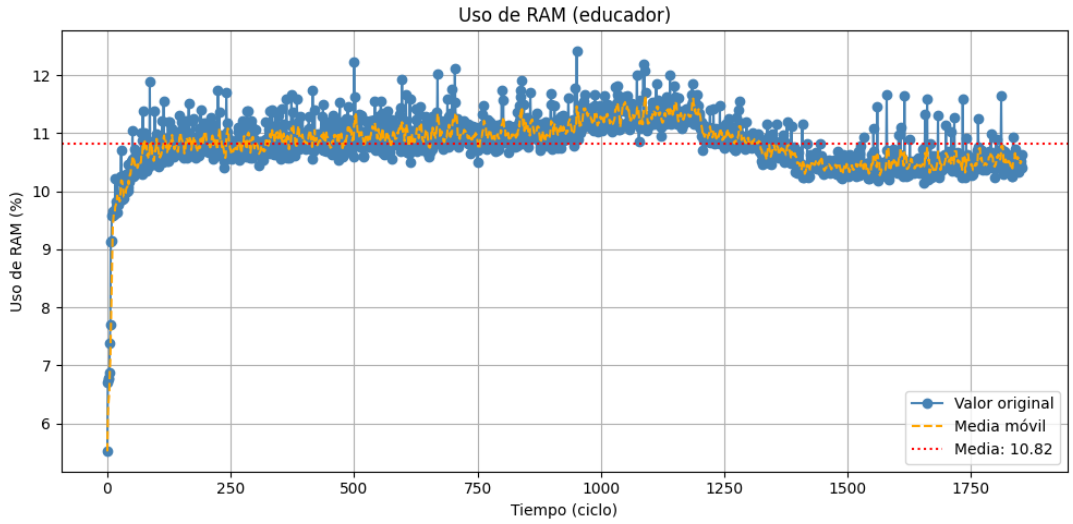

Figura 5.9: Uso de RAM durante la sincronización inicial (perfil educador, el más bajo).

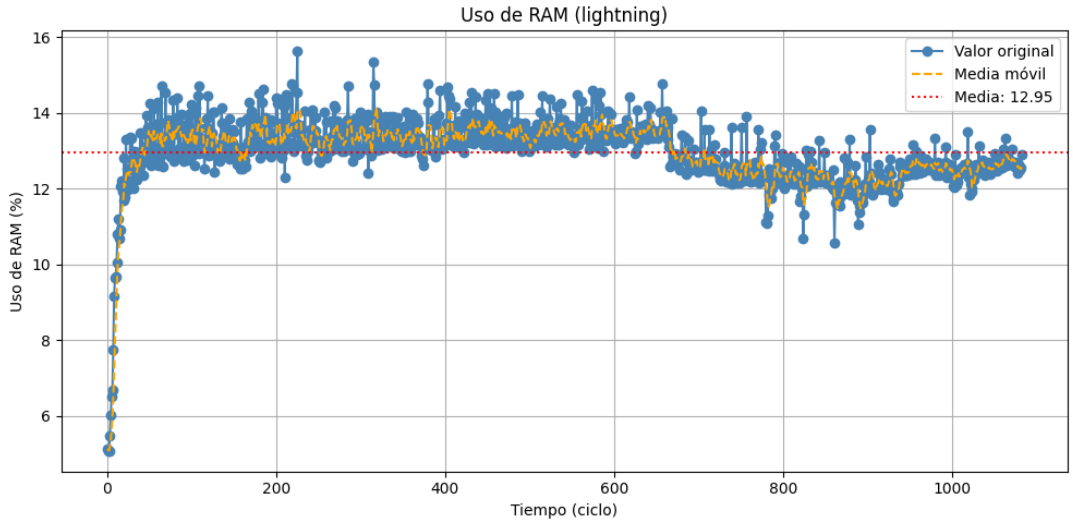

Figura 5.10: Uso de RAM durante la sincronización inicial (perfil lightning, el más alto).

Uso de disco durante la sincronización

La comparativa a nivel general (Fig. 5.11) muestra una clara división entre dos patrones. Los perfiles **sin podar** (**Estándar**, **Analista**, **Validador**) muestran resultados en torno a 520-560 GB, lo cual se puede interpretar como el almacenamiento completo de la cadena. El **Validador** (Fig. 5.13) sería claramente el perfil más exigente, alcanzando cifras superiores a las de Estándar y Analista, ya que incluye configuraciones correspondientes a índices y el estado UTXO. Por otro lado, los perfiles **podados** (**dApps**, **Educador**, **Lightning**) se distribuyen en cifras que abarcan entre los 30-35 GB. El **dApps** (Fig. 5.14) es el que

menos espacio requiere, confirmando que realizar un proceso de poda dará lugar a un impacto en el disco con cifras tan bajas como las mencionadas. El **Estándar** (Fig. 5.12) funcionaría como base, mostrando una trayectoria lineal de crecimiento continuo, mientras que los perfiles ligeros muestran mesetas al superar la línea de umbral de poda. Esto representa una forma de hacer ver que el disco es la variable que más claramente diferencia los perfiles de usuario: unos requieren hardware bien robustos para poder almacenar datos, mientras que otros se podrían ejecutar incluso en equipos con muy poca capacidad.

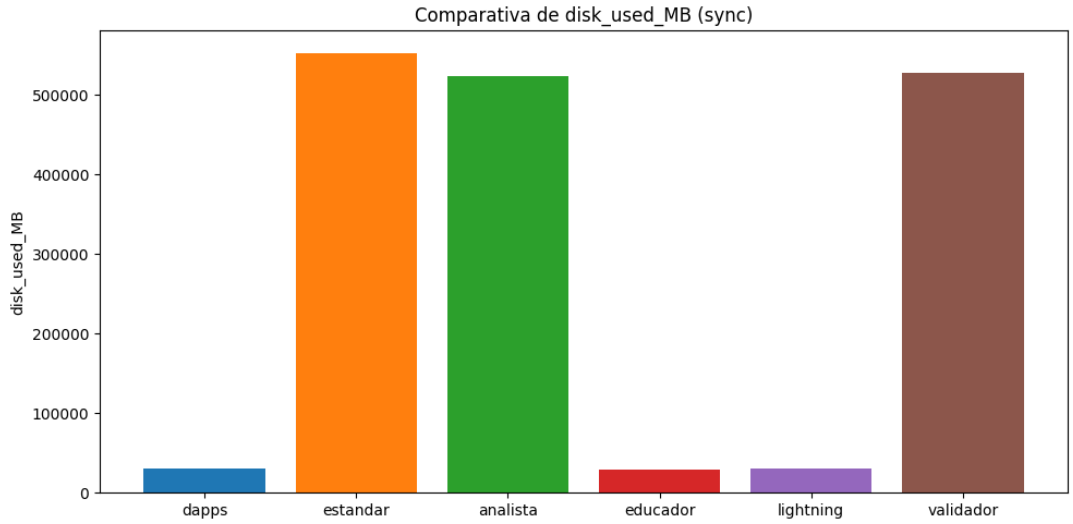

Figura 5.11: Uso de disco durante la sincronización por perfil.

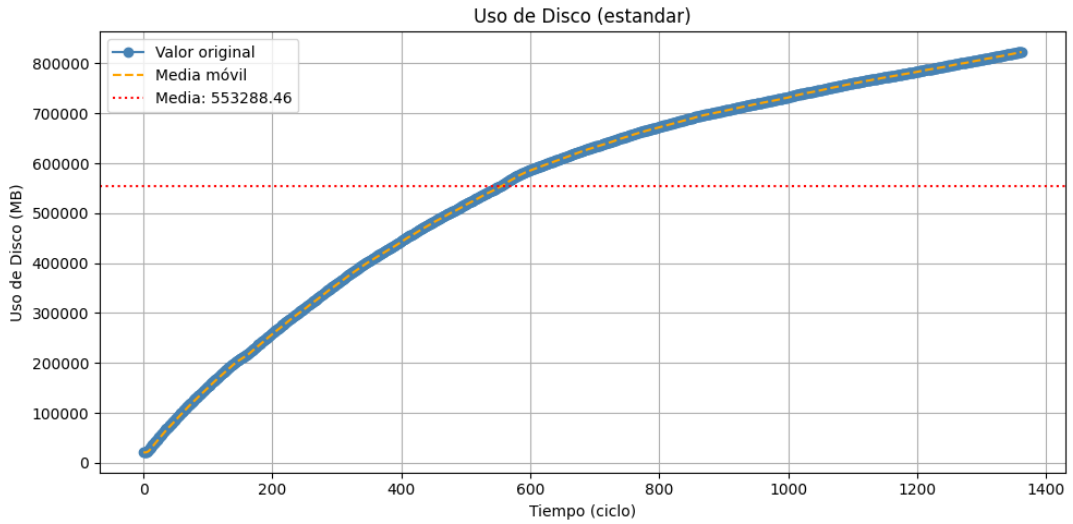

Figura 5.12: Crecimiento del uso de disco durante la sincronización (perfil Estándar).

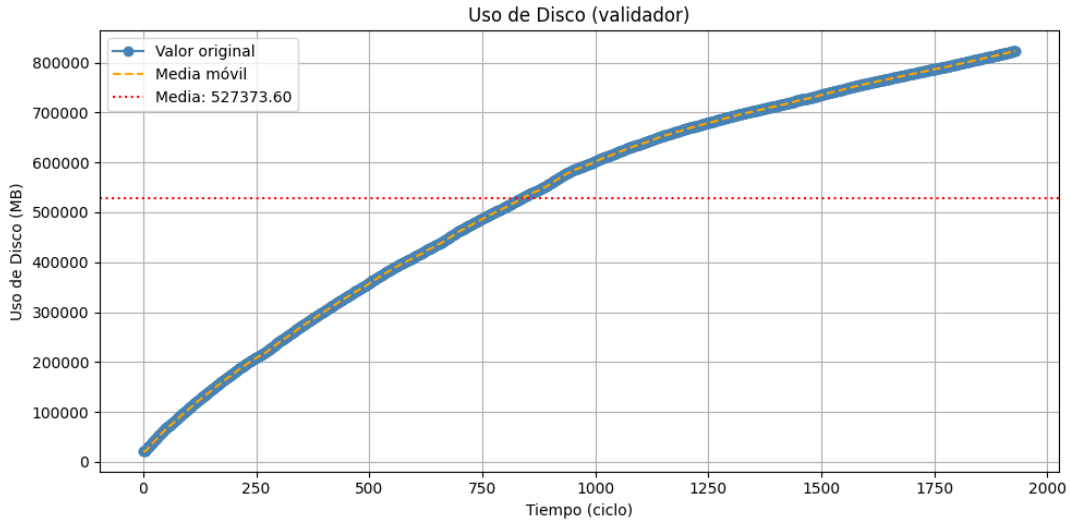

Figura 5.13: Crecimiento del uso de disco durante la sincronización del perfil Validador, el más alto (sin considerar perfil Estándar).

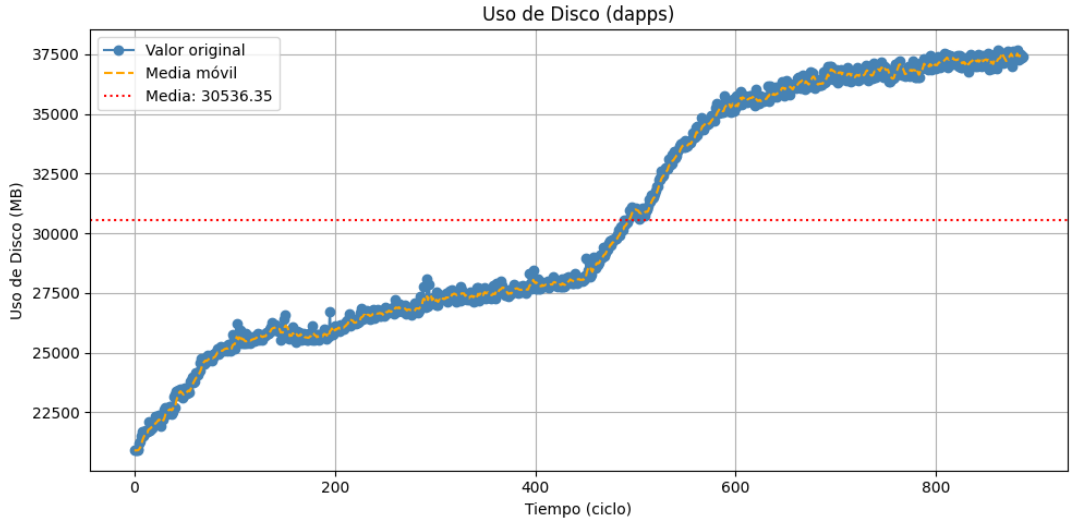

Figura 5.14: Crecimiento del uso de disco durante la sincronización (perfil dApps, el más bajo).

Tráfico de red durante la sincronización

El análisis del tráfico de red evidencia un comportamiento **muy homogéneo entre los perfiles** en términos de volumen total recibido, con valores que oscilan en torno a los $6{,}8 \times 10^8$ kB (Fig. 5.15). Esto se explica porque los registros de `net_rx_kB` representan **valores acumulativos del sistema operativo desde el arranque**, y no un flujo instantáneo.

No obstante, al observar las curvas temporales individuales (Figuras 5.16 a 5.18), se aprecian **diferencias en la intensidad del tráfico por intervalo**: los perfiles **dApps** y **Lightning** presentan picos iniciales más pronunciados, seguidos de descensos graduales, mientras que los perfiles **Validador**, **Analista** y **Estándar** mantienen un comportamiento más sostenido y regular durante toda la sincronización.

En el tráfico de salida (Figuras 5.19 a 5.22), las diferencias son mucho más acentuadas. El perfil **Educador**, por ejemplo, transporta aproximadamente $3{,}5 \times 10^6$ kB de tráfico, siendo el perfil que más tráfico envía. Seguidos por el perfil **Analista** junto con el perfil **Estándar** que obtienen resultados parecidos, mientras que el comportamiento de **Lightning** y **dApps** muestra valores intermedios y el **Validador** es el que aporta menos tráfico. La situación refleja (como ya se había comentado en el tráfico de entrada) que los perfiles de demostración o análisis (como el **Educador** o el **Analista**) son perfiles que tienden a transmitir un mayor flujo de tráfico a través de la red P2P, mientras que los perfiles de validación estricta favorecen la localización y el procesamiento del tráfico en la validación de bloques, entre otros. Por lo tanto, también se puede decir que, de manera general, los perfiles ligeros no tienden a recortar el tráfico total, pero sí que muestran una **mayor variabilidad** en el flujo por intervalo. Para llevar a cabo una buena interpretación, hay que tener en cuenta que:

> Las acumulaciones de registros de red (`net_rx_kB`, `net_tx_kB`) son los valores acumulados[3] que el sistema operativo ha ido reportando en el intervalo de tiempo comprendido entre el arranque del mismo y el término de este. Por este motivo, los gráficos comparativos mostrarán el tráfico de forma acumulativa y no instantánea, tal como puede aparecer en intervalos.

[3] En Linux, los contadores de red acumulativos se encuentran en `/sys/class/net/[iface]/statistics/`.

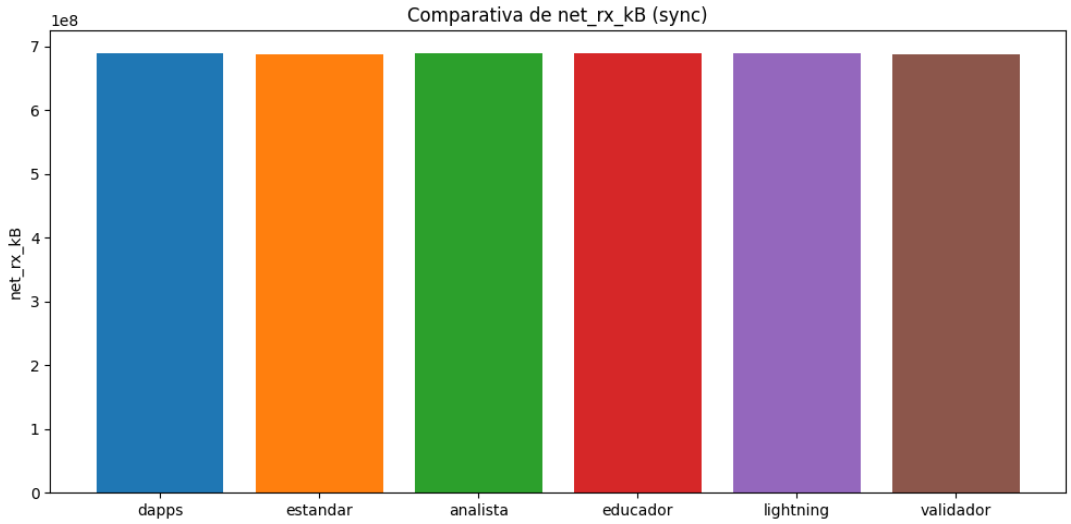

Figura 5.15: Tráfico de red entrante durante la sincronización por perfil.

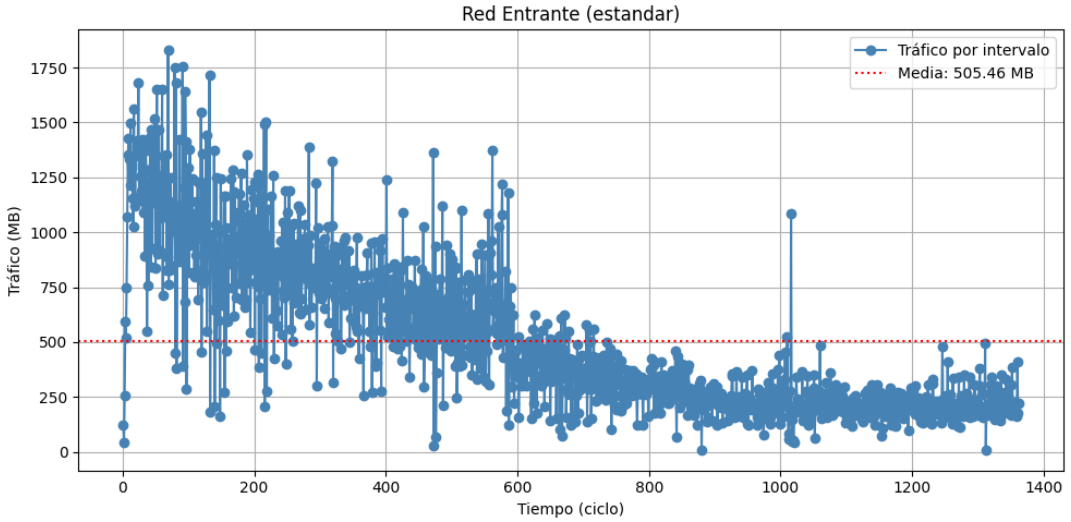

Figura 5.16: Tráfico de red entrante durante la sincronización (perfil Estándar).

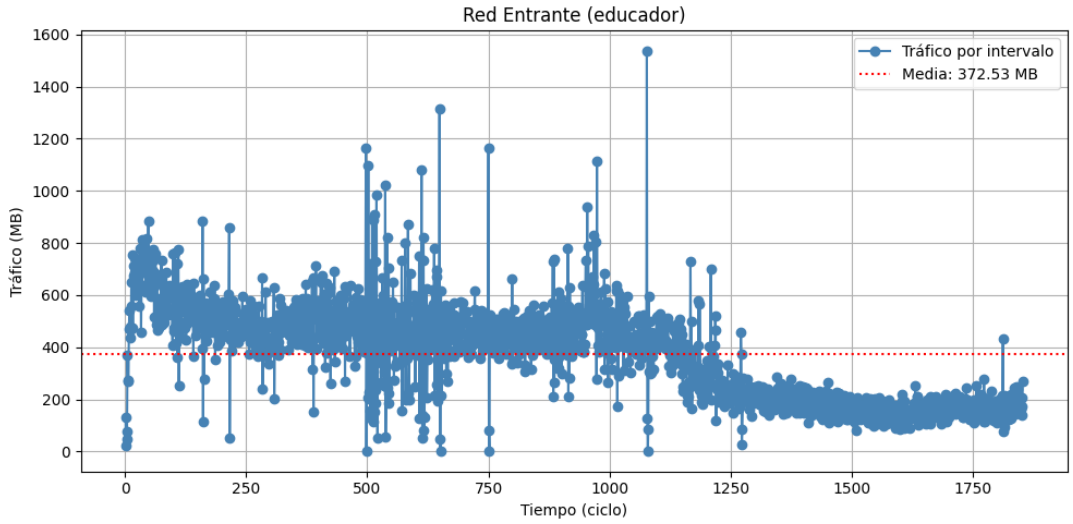

Figura 5.17: Tráfico de red entrante durante la sincronización (perfil Educador).

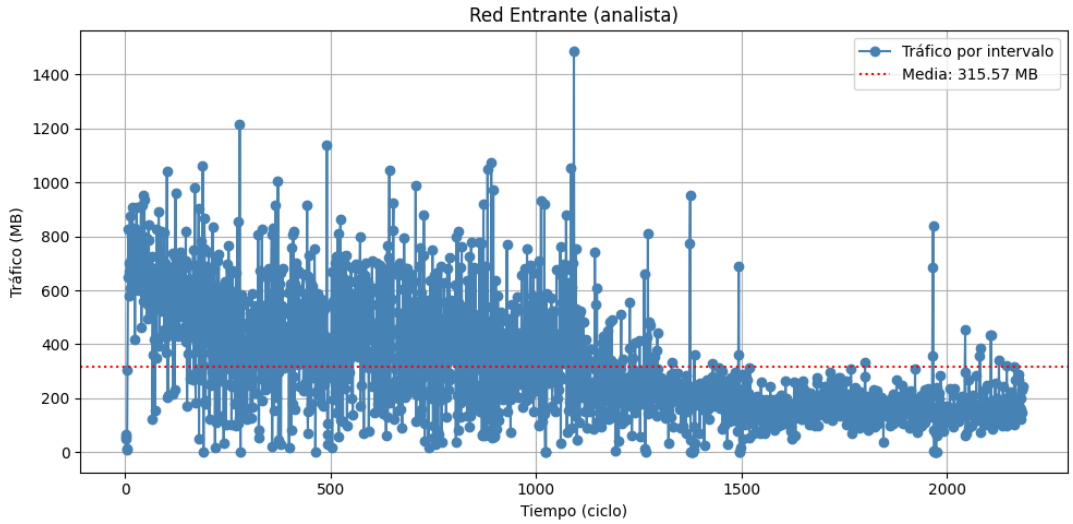

Figura 5.18: Tráfico de red entrante durante la sincronización (perfil Analista).

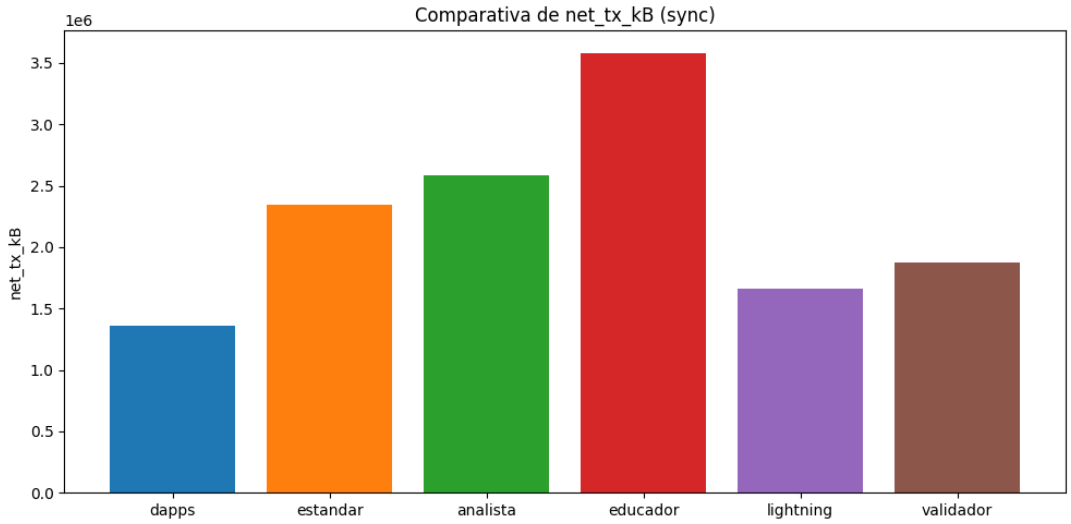

Figura 5.19: Tráfico de red saliente durante la sincronización por perfil.

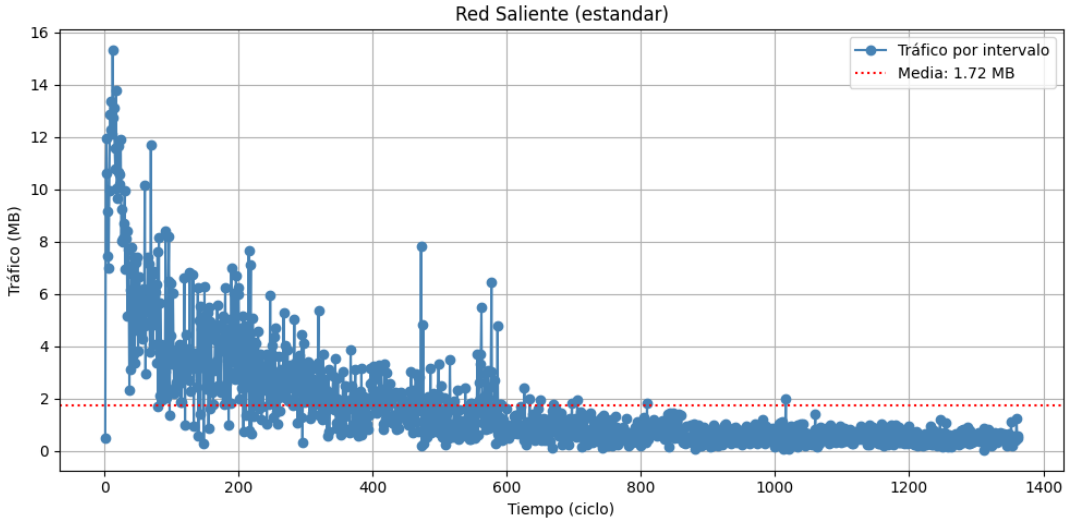

Figura 5.20: Tráfico de red saliente durante la sincronización (perfil Estándar).

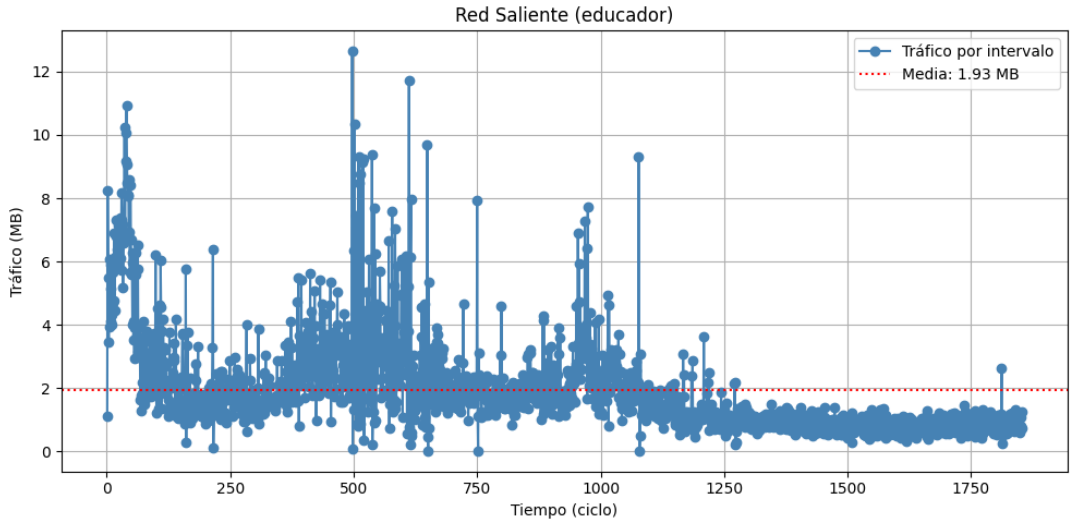

Figura 5.21: Tráfico de red saliente durante la sincronización (perfil Educador, el más alto).

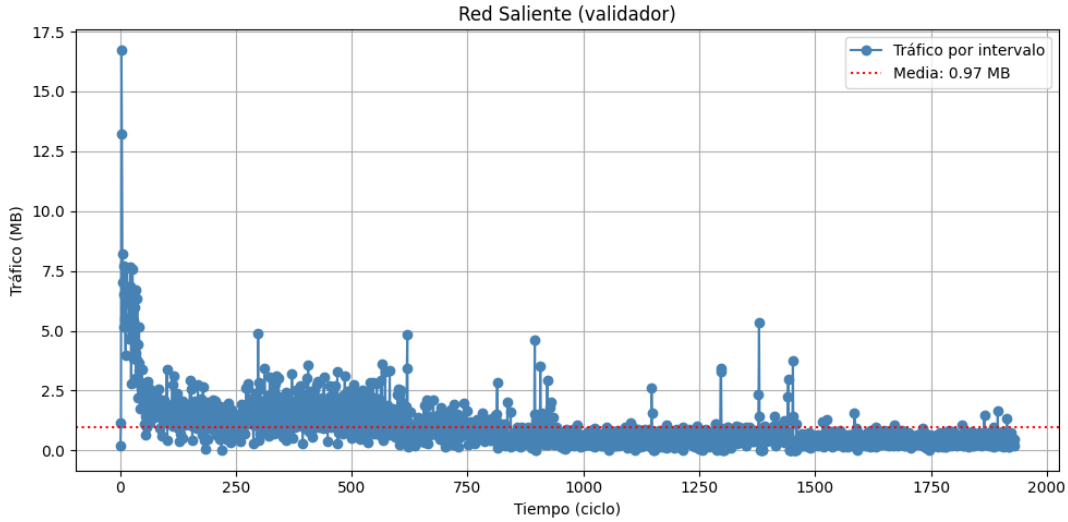

Figura 5.22: Tráfico de red saliente durante la sincronización (perfil Validador, el más bajo).

Progreso de verificación durante la sincronización

El indicador `verificationprogress` refleja la velocidad a la cual cada perfil alcanza el 100 % de sincronización (o en la práctica el valor más cercano al 100 %). El **perfil Estándar** (Fig. 5.24) actúa como referencia, con una curva regular que tiene un avance estable hasta el 100 %. El **perfil Educador** (Fig. 5.25) termina el proceso en tiempos más cortos gracias al pruning y a su menor carga de disco, aunque con valores medios

de velocidad de sincronización algo menores en la comparativa global (Fig. 5.23). El **perfil Validador** (Fig. 5.26), en cambio, avanza a una velocidad menor debido a la validación profunda de todos los bloques y por lo mismo, se sitúa en la parte baja de la distribución. De esta manera, los perfiles ligeros **(Educador, dApps, Lightning)** presentan un progreso más rápido, por el contrario, los perfiles de alta carga de validación **(Validador y Analista)** tardan más en llegar al final de la IBD, lo cual reafirma la relación causa-efecto entre la intensidad de verificación y la velocidad de sincronización.

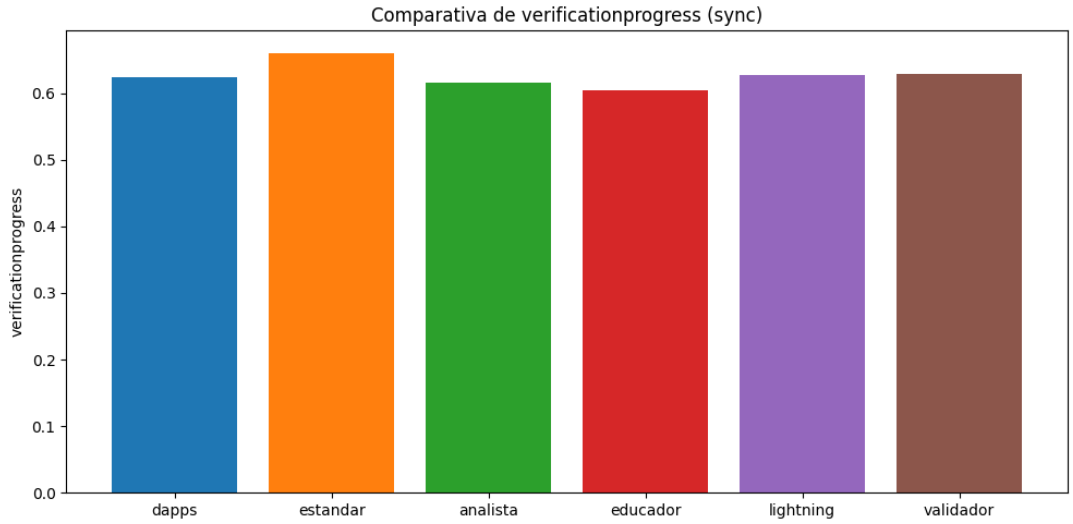

Figura 5.23: Progreso de verificación (`verificationprogress`) durante la sincronización por perfil.

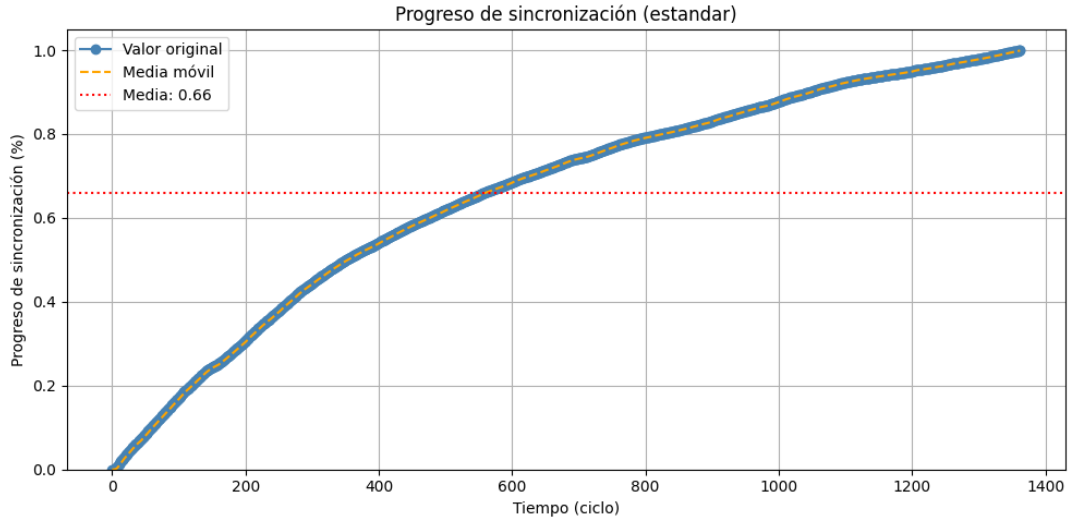

Figura 5.24: Progreso de verificación durante la sincronización (perfil Estándar).

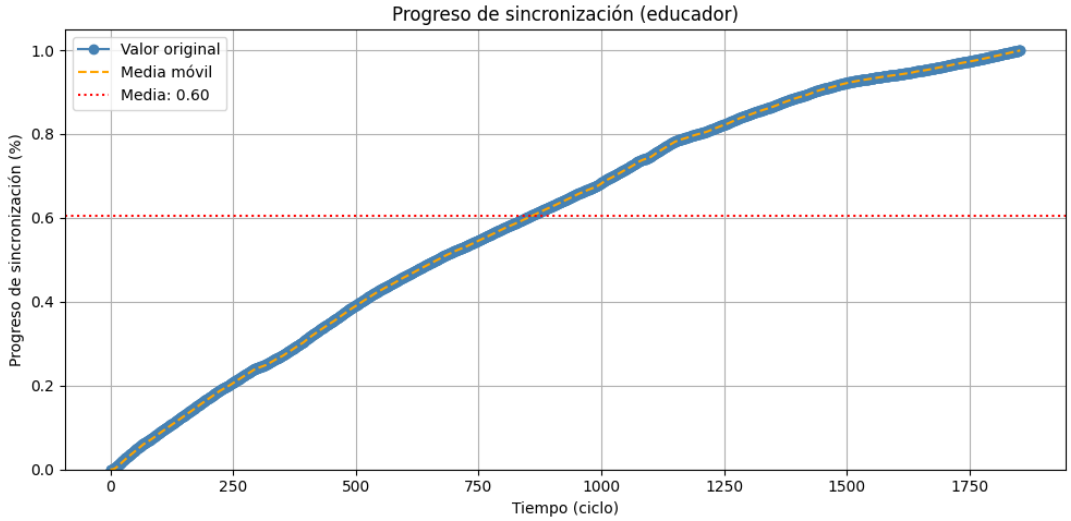

Figura 5.25: Progreso de verificación durante la sincronización (perfil Educador).

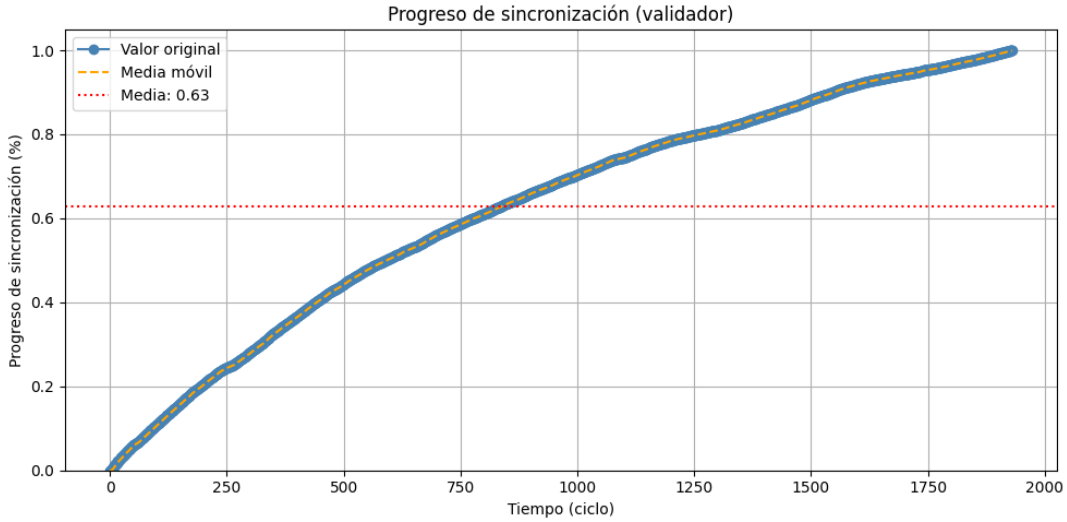

Figura 5.26: Progreso de verificación durante la sincronización (perfil Validador).

Resumen comparativo de la fase de sincronización

La media de las métricas principales se resume en la Tabla 5.3 (valores normalizados manualmente a partir del CSV). Posteriormente, la Tabla 5.4 representa de forma visual los perfiles que dan mejor resultado relativo para cada parámetro. El análisis general indica que los perfiles **Validador** y **Analista** son los más intensivos en lo que se refiere a uso de disco y memoria, mientras que el perfil **Educador** se distingue por su bajo uso de RAM y disco, a costa del elevado uso de CPU y red. El rendimiento del perfil **dApps** se distingue

Métrica	Educador	Lightning	Analista	dApps	Estándar	Validador
CPU (%)	54,1	37,6	38,1	37,2	37,7	38,4
RAM (%)	10,8	13,0	12,6	12,5	13,0	12,7
Disco (GB)	29,7	30,2	524,0	30,5	553,3	527,4
Red RX (GB)	690,29	690,51	689,20	689,71	688,44	688,61
Red TX (GB)	3,58	1,66	2,59	1,36	2,35	1,87
Verificationprogress	0,60	0,63	0,62	0,62	0,66	0,63

Tabla 5.3: Valores medios durante la sincronización inicial (métricas acumulativas/porcentuales).

notoriamente por su equilibrio entre CPU y disco. Y, por su parte, el perfil **Estándar** destaca con los resultados más altos en la verificación, siendo el soporte de referencia base.

La actualización de los resultados de sincronización confirma que los perfiles de menor peso mantienen la tendencia esperada: el perfil **Educador** sigue siendo el mejor perfil en lo que se refiere al uso de RAM y disco, pero es intensivo en el tráfico saliente (TX). El perfil **Lightning** llega a un tráfico entrante ligeramente más alto que los demás perfiles, en coherencia con su rol como backend de los canales de pago, mientras que el perfil **dApps** sigue presentando el menor consumo de CPU, manteniendo además la estabilidad en entornos de desarrollo. El perfil **Estándar** sigue siendo el punto medio de referencia y el perfil **Validador** sigue siendo el que representa la mayor solidez en la verificación completa[4]. En general, los datos enfatizan que el rendimiento relativo de cada uno de los perfiles está en consonancia con el objetivo funcional de su ajuste en la fase intensiva de la misma sincronización.

Proceso de post-sincronización

Uso de CPU en post-sincronización

En la etapa de la post-sincronización, el uso de CPU se estabiliza significativamente respecto al proceso de sincronización de arranque. En la Figura 5.27 se muestra la comparativa del uso medio de CPU entre diferentes perfiles, los cuales revelan que el perfil `dApps` tiene el mayor uso medio (7.97 %), mientras que el perfil `Validador` es el que menos lo utiliza (4.72 %), y el perfil `Estándar` (que sirve como línea base), aparece con un valor intermedio (5.59 %).

En los datos evolutivos de la temporalidad (Figuras 5.28, 5.29 y 5.30), se puede afirmar que, tras unos picos iniciales, todos los perfiles acaban convergiendo en el uso de CPU,

[4] "Verificación completa" alude al estado en que el nodo ha validado toda la cadena (initialblockdownload=false). Los valores de verificationprogress en las tablas son promedios de la fase inicial y, por tanto, inferiores a 1.

Métrica	Educador	Lightning	Analista	dApps	Estándar	Validador
CPU (menor es mejor)				✓		
RAM (menor es mejor)	✓					
Disco (menor es mejor)	✓					
Red RX (mayor es mejor)		✓				
Red TX (mayor es mejor)	✓					
Verificationprogress (mayor es mejor)					✓	

Tabla 5.4: Perfiles con mejor rendimiento relativo durante la sincronización inicial

que se mantiene a un nivel bajo y estable, confirmando así que la carga computacional en un régimen estable es muy inferior a la encontrada durante la fase de sincronización de arranque, sin importar el perfil.

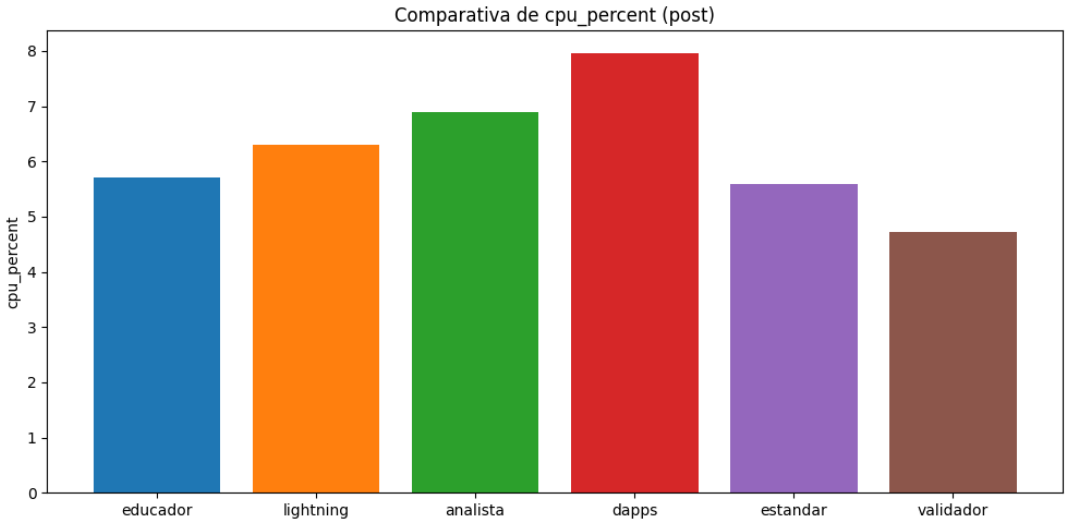

Figura 5.27: Comparativa de uso de CPU promedio en post-sincronización por perfil.

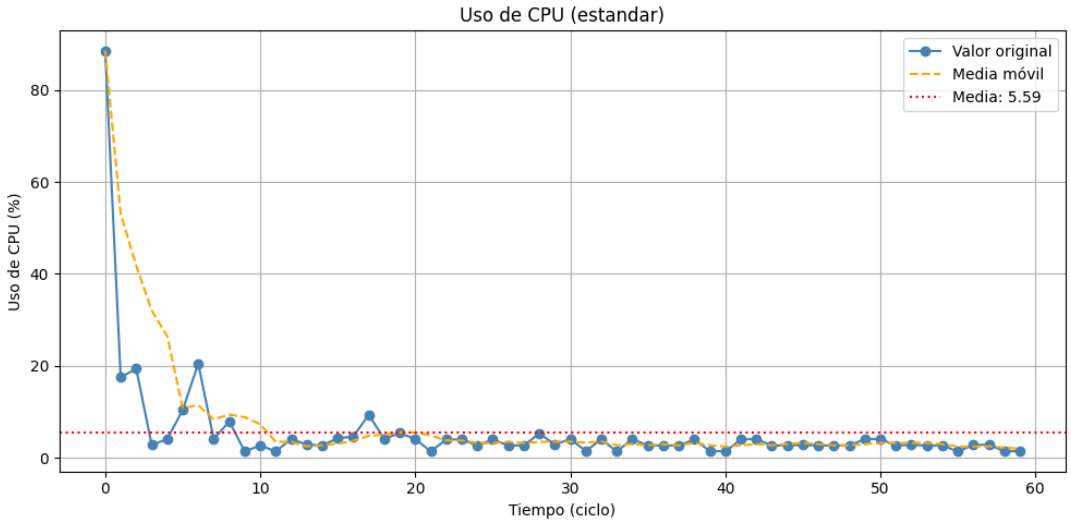

Figura 5.28: Uso de CPU en post-sincronización (perfil Estándar).

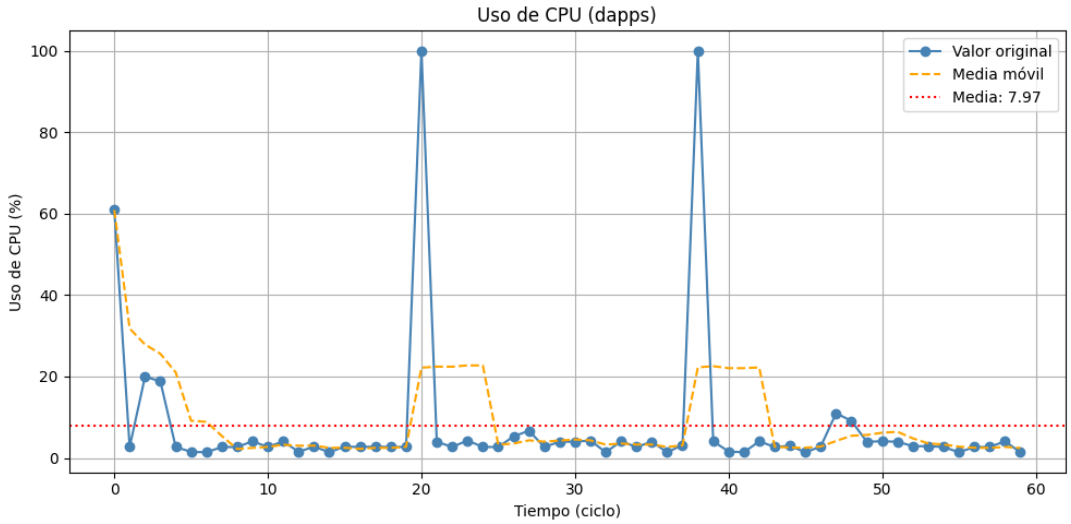

Figura 5.29: Uso de CPU en post-sincronización (perfil dApps).

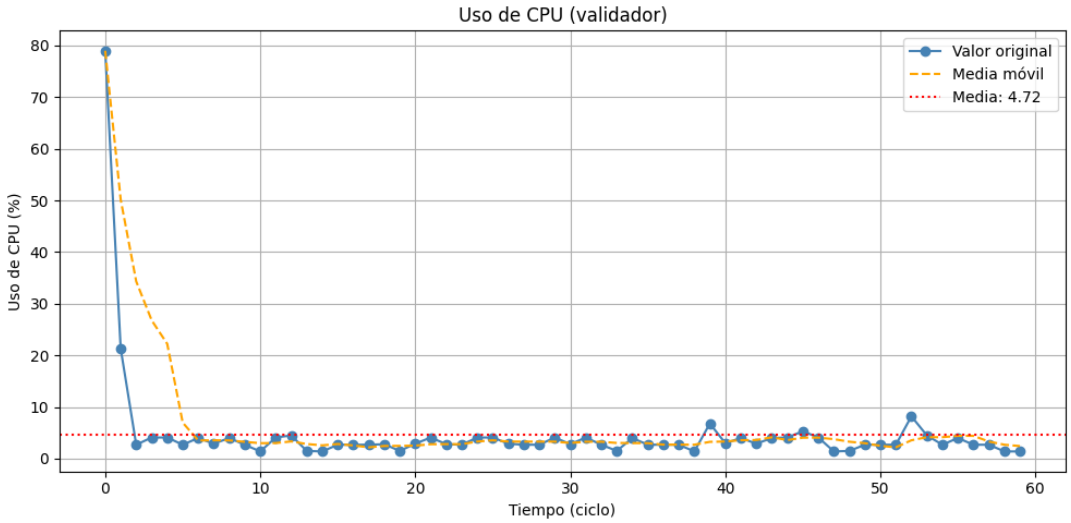

Figura 5.30: Uso de CPU en post-sincronización (perfil Validador).

Uso de RAM en post-sincronización

Durante la fase de post-sincronización, el consumo de memoria RAM se mantiene bastante estable y con oscilaciones notablemente más limitadas en comparación con la sincronización inicial. En la Figura 5.31, se puede observar cómo queda la comparativa del uso medio de RAM a través de los diferentes perfiles, en donde, por ejemplo, el perfil `Estándar` tiene un uso medio en RAM más elevado (1941 MB), mientras que el perfil `Educador` se encuentra con un uso de RAM muy por debajo (1543 MB). Estos valores contrastan con la línea base, que se mantiene como un punto intermedio de referencia.

La evolución temporal (Figuras 5.32 a 5.33) muestra cómo todos los perfiles tienen un breve ajuste inicial con los picos en los primeros ciclos, seguidos de valores prácticamente constantes. El resultado es que, cuando la IBD ha finalizado, el nodo que realiza la post-sincronización no necesita incrementar en gran medida el uso de la RAM, sea el perfil que sea.

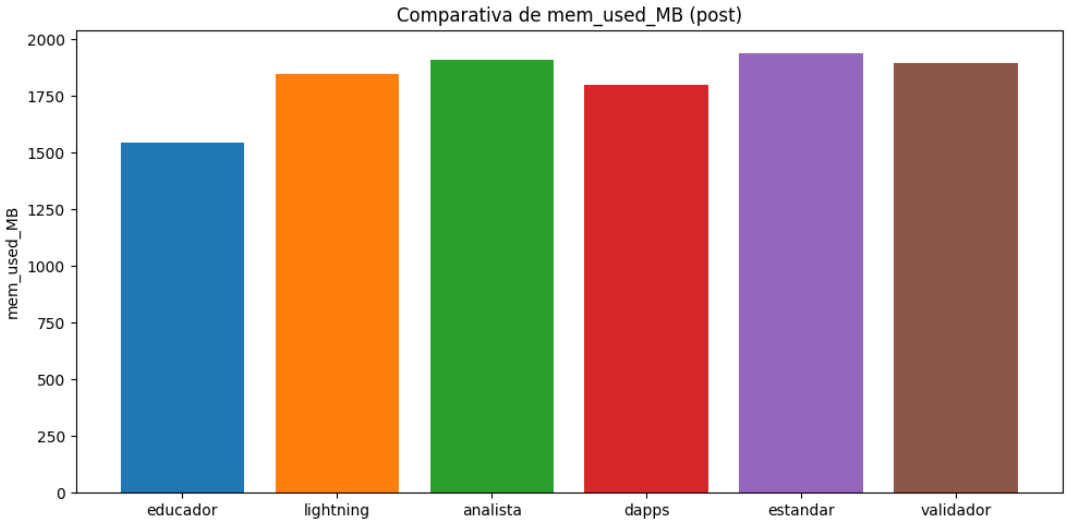

Figura 5.31: Comparativa de uso de RAM promedio en post-sincronización por perfil.

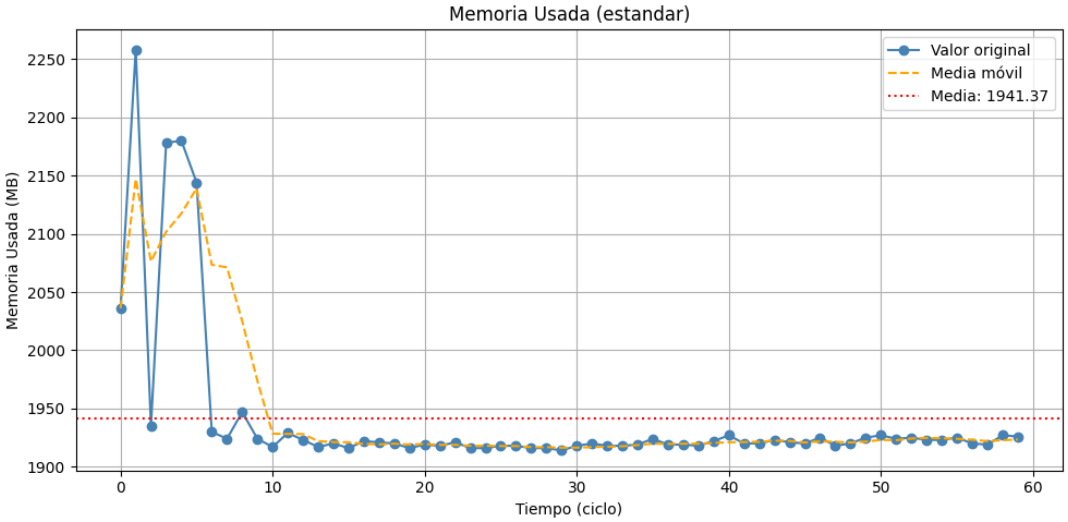

Figura 5.32: Uso de RAM en post-sincronización (perfil Estándar).

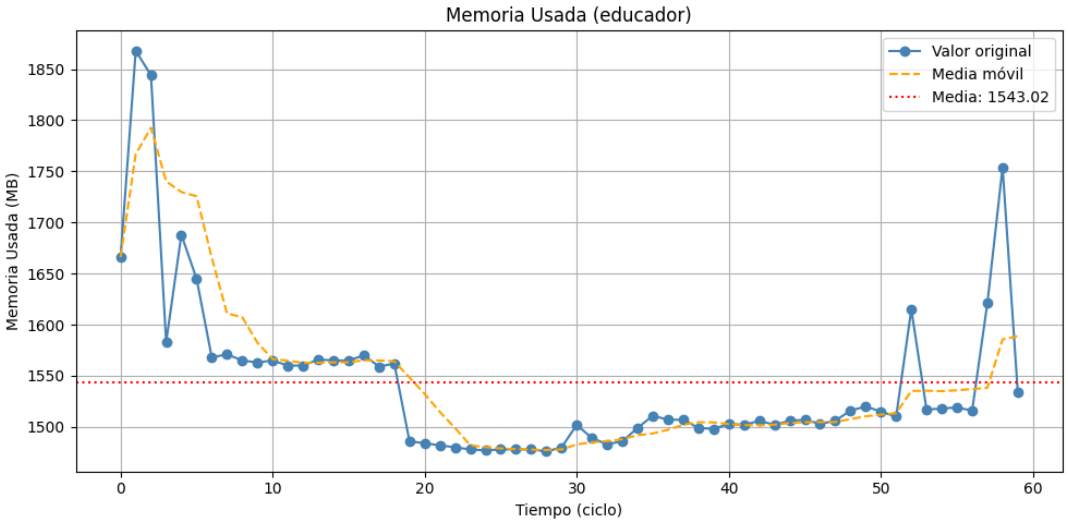

Figura 5.33: Uso de RAM en post-sincronización (perfil Educador).

Uso de disco en post-sincronización

Se presenta una clara estabilización en el uso de disco de acuerdo a sus cifras en la fase de post-sincronización respecto al proceso de sincronización inicial. En particular, la Figura 5.34 muestra que los perfiles `Analista`, `Estándar` y `Validador` muestran el mayor uso respecto al espacio en disco (por encima de los 820.000 MB), mientras que los perfiles `dApps`, `Educador` y `Lightning` los que muestran consumos mucho más bajos (cercanos a los 35.000 MB).

El perfil `Estándar`, que se utiliza como línea base, presenta un uso estable por encima de los 823.301 MB (Figura 5.35). El perfil `Analista` muestra un valor ligeramente más alto (835.816 MB; Figura 5.36) y finalmente el perfil textttdApps muestra un uso muy bajo (37.105 MB; Figura 5.37). Esta diferencia viene a dejar bien claro el impacto que puede llegar a tener la configuración del perfil en cuanto al uso del espacio en disco, y que debe ser considerado a la hora de dimensionar los recursos de infraestructura en función de cada caso de uso.

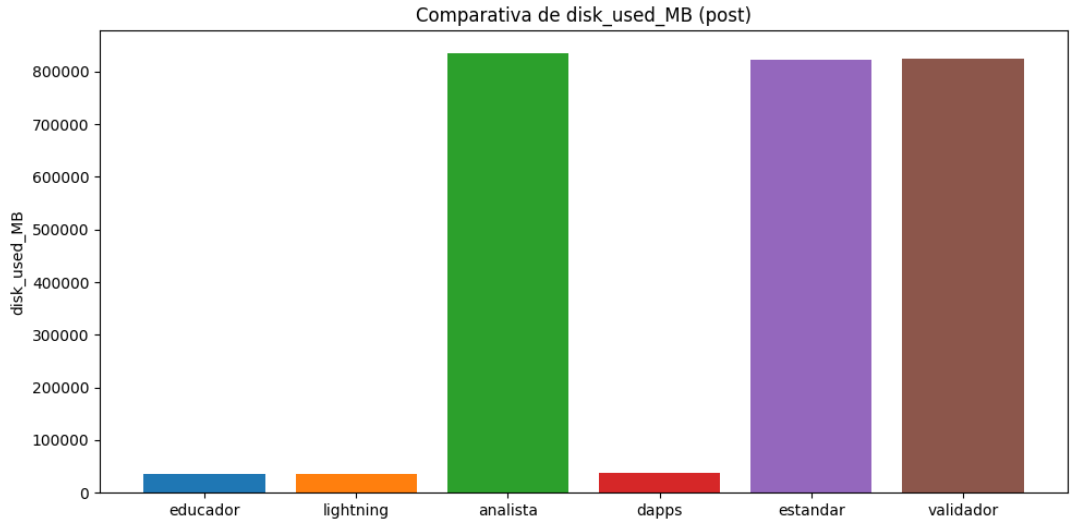

Figura 5.34: Comparativa de uso de disco en post-sincronización por perfil.

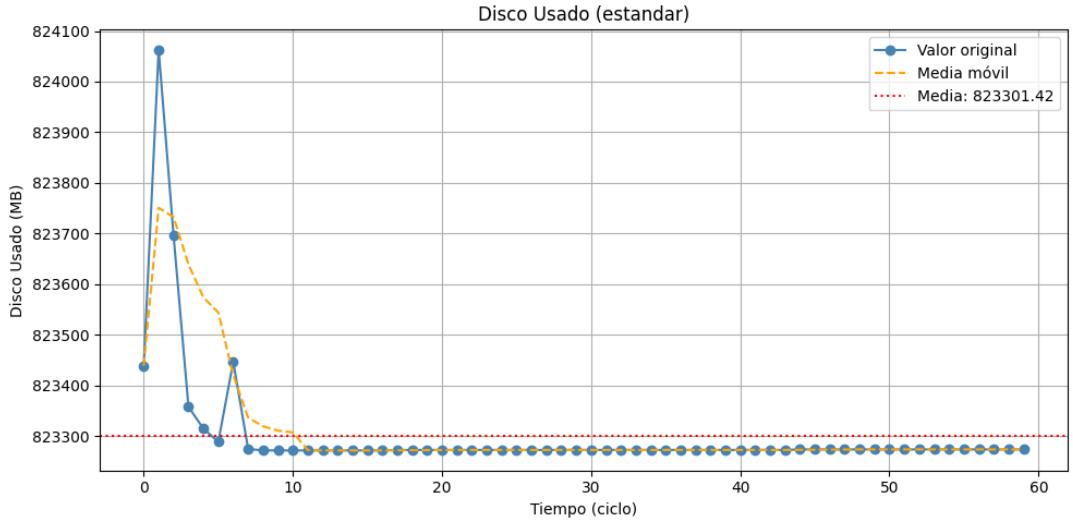

Figura 5.35: Uso de disco en post-sincronización (perfil Estándar).

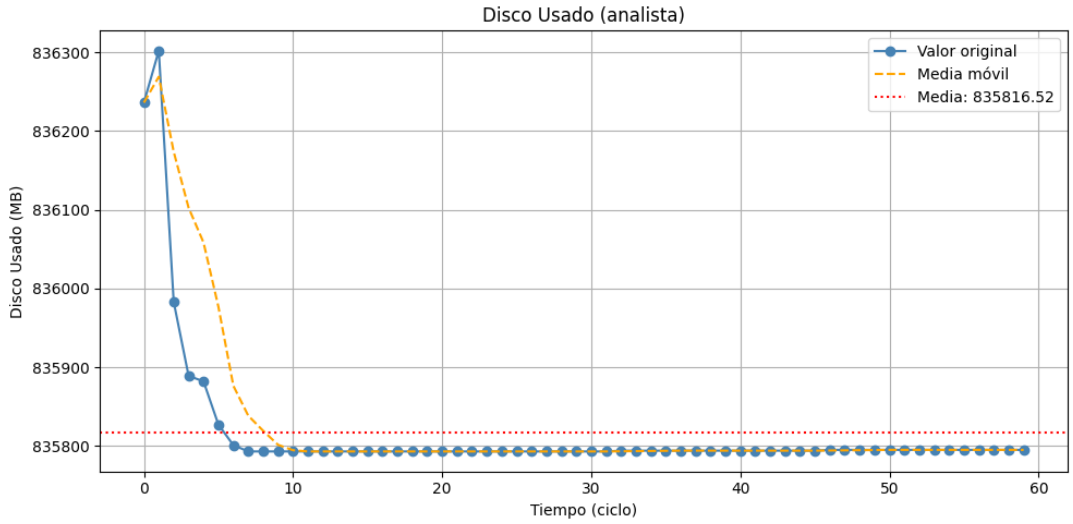

Figura 5.36: Uso de disco en post-sincronización (perfil Analista).

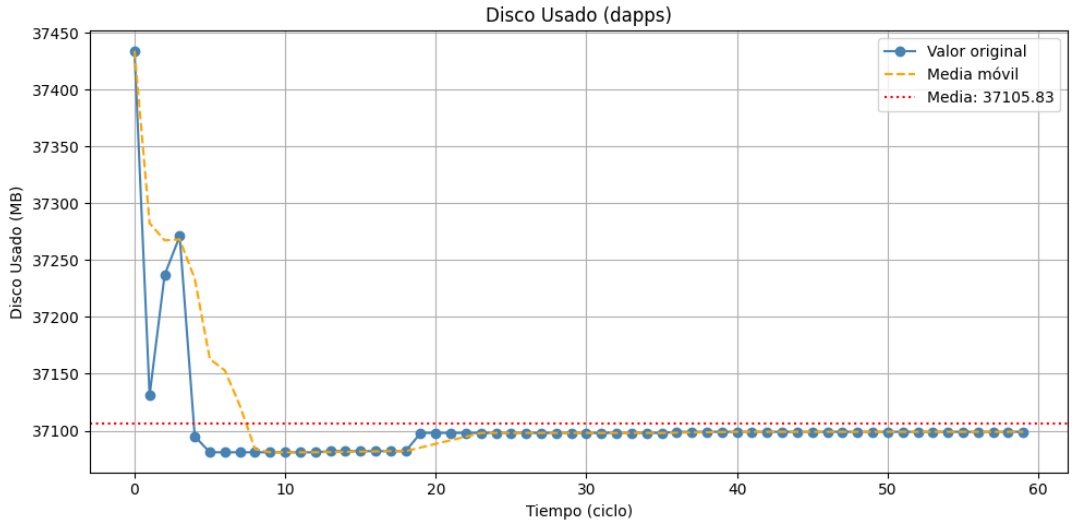

Figura 5.37: Uso de disco en post-sincronización (perfil dApps).

Tráfico de red en post-sincronización

Antes de analizar las métricas de esta fase, conviene aclarar la metodología empleada: a diferencia de la etapa de sincronización inicial, en la cual los registros de red (`net_rx_kB`, `net_tx_kB`) corresponden a valores acumulativos reportados por el sistema operativo, en la fase de *post-sincronización* las cifras reflejan la **suma o media de las variaciones**

incrementales (deltas)[5] observadas durante una ventana temporal de 60 minutos, con un muestreo cada 60 segundos.

En consecuencia, los volúmenes de tráfico representados son **varios órdenes de magnitud menores** que los de la IBD, pues se registran únicamente los flujos instantáneos de transmisión y recepción, y no el crecimiento total acumulado desde el arranque.

Para asegurar la comparabilidad entre fases y perfiles, los archivos `.csv` fueron previamente depurados: se realizó conversión de unidades, eliminación de registros vacíos y **normalización manual de las métricas** (CPU, RAM, disco y red) de modo que todas las tablas reflejaran valores medios homogéneos y comparables.

Cabe indicar que, en el momento de la primera sincronización (IBD), las métricas de memoria se presentan en porcentajes, y tras la sincronización en unidades absolutas (MB), de forma que se refleje más tangible el consumo real del sistema.

Durante la etapa de post-sincronización, los valores de la red se mantuvieron **muy por debajo de los observados durante la IBD**, lo que indicaba que el nodo había pasado a un estado diferente y estable. En la comparativa general (Figuras 5.38 a 5.40), los perfiles **Lightning** y **Educador** presentan los mayores volúmenes de tráfico entrante (aproximadamente $7{,}5 \times 10^4$ kB); les siguen en la comparativa **Analista** y **Estándar** (entre 6,4 y $6{,}8 \times 10^4$ kB), mientras que **Validador** y **dApps** presentan los niveles más bajos. Este comportamiento se relaciona con la propia intención de cada perfil, ya que los perfiles orientados a red (Lightning y Educador) mantienen conexiones más activas, a diferencia de los perfiles de validación o demostración, los cuales cuentan con menos procesamiento de los mensajes P2P.

En lo que respecta al tráfico de salida (Figuras 5.41 a 5.43), el perfil **dApps** se convierte en el que dispone mayor cantidad de tráfico (aproximadamente $7{,}6 \times 10^3$ kB), seguido muy de cerca por **Validador** y **Lightning**. Los perfiles **Educador** y **Estándar** muestran valores más reducidos, mientras el **Analista** se queda en valores medios. Estos resultados muestran que los perfiles con interacción externa (**dApps** y **Lightning**) conservan un mayor tráfico de salida, de acuerdo con su función en entornos de desarrollo o en sistemas de pago.

En definitiva, la fase posterior a la sincronización indica que **el tráfico de la red se ha consolidado** de forma que se han obtenido valores bajos y diferencias bajas entre los perfiles. Los patrones en sí son reflejo de la naturaleza de cada uno de ellos: los perfiles **Lightning** y **dApps** son los que más destacan y constituyen el referente de comunicación continua; por otro lado, el perfil **Educador** muestra un flujo elevado que nos hace suponer que coincide con su uso en entornos pedagógicos o de prueba.

[5] En este contexto, Δ indica la variación medida entre dos puntos de muestreo consecutivos en intervalos de 60 s.

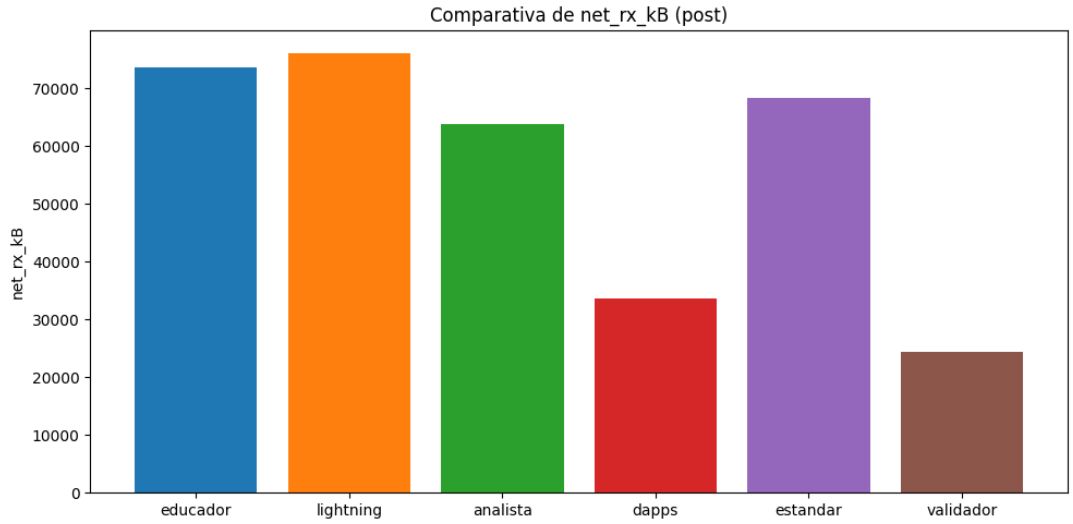

Figura 5.38: Comparativa del tráfico de red entrante (`net_rx_kB`) en post-sincronización por perfil.

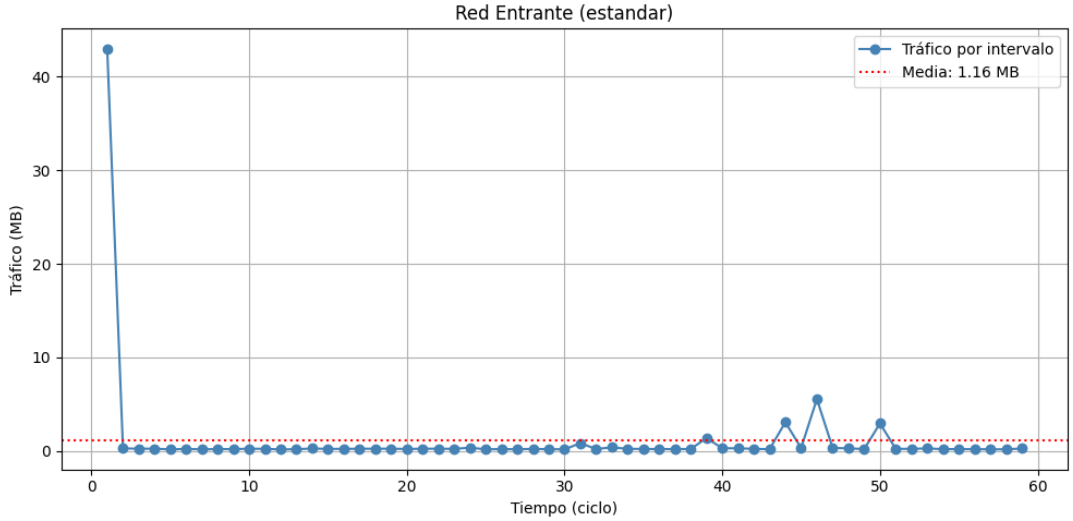

Figura 5.39: Tráfico de red entrante en post-sincronización (perfil Estándar).

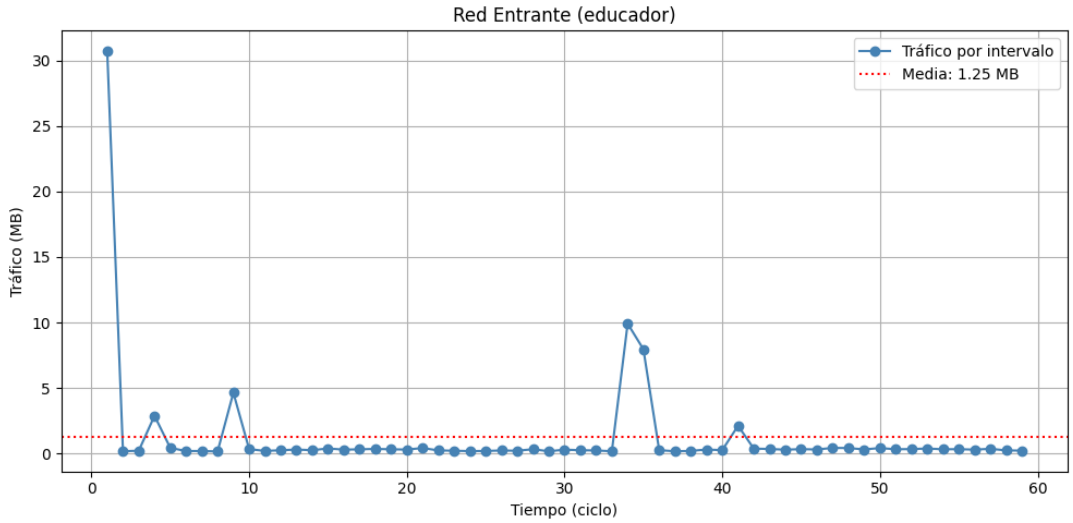

Figura 5.40: Tráfico de red entrante en post-sincronización (perfil Educador).

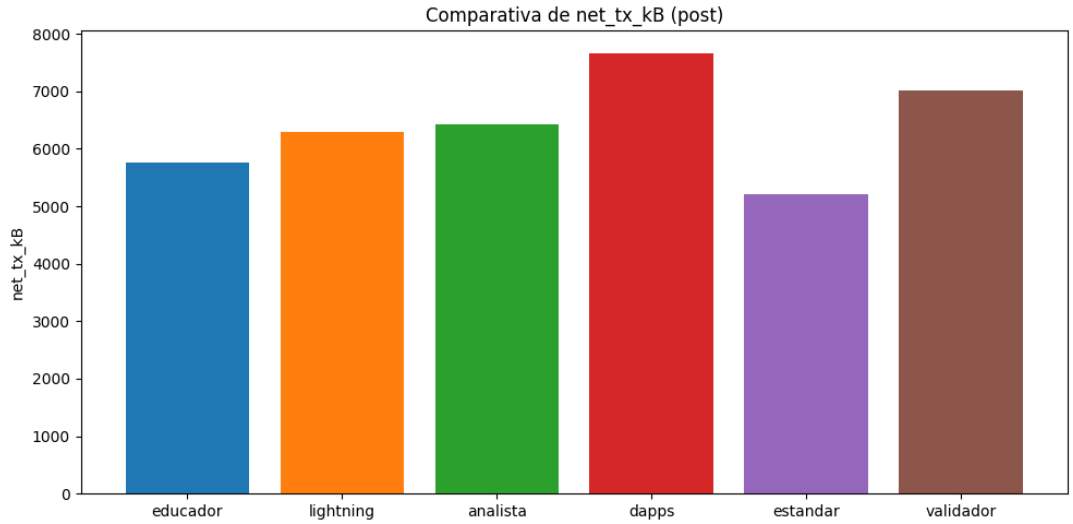

Figura 5.41: Comparativa del tráfico de red saliente (`net_tx_kB`) en post-sincronización por perfil.

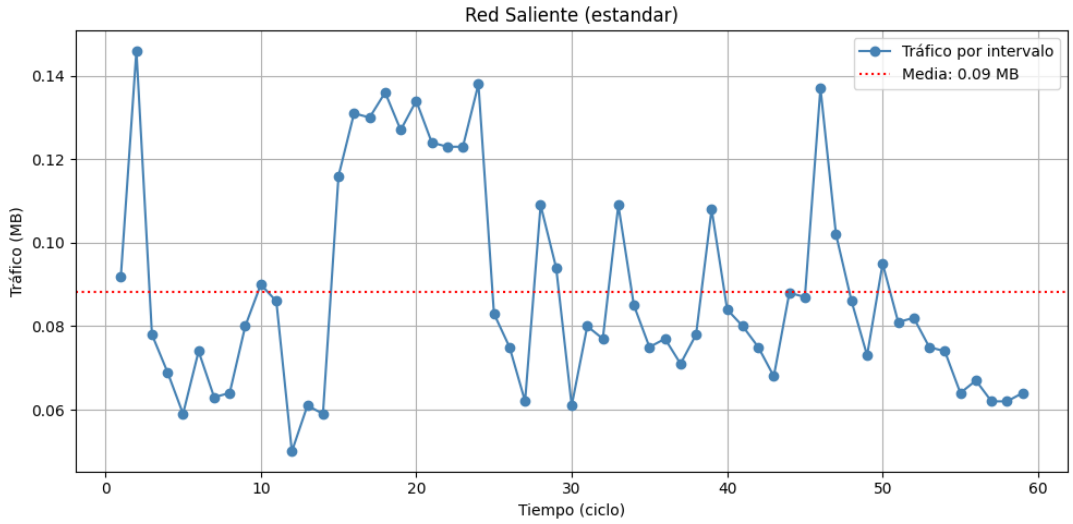

Figura 5.42: Tráfico de red saliente en post-sincronización (perfil Estándar).

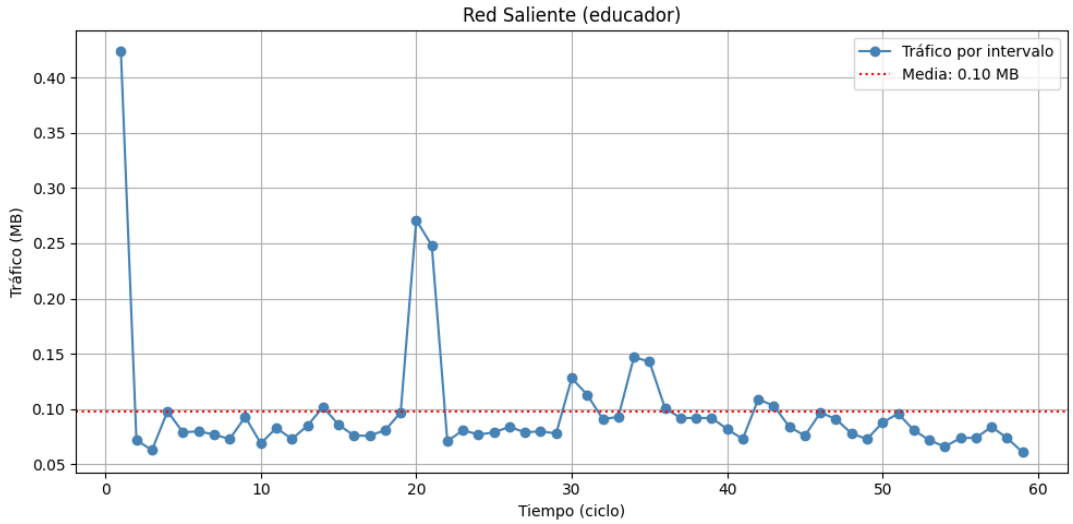

Figura 5.43: Tráfico de red saliente en post-sincronización (perfil Educador).

Número de procesos en post-sincronización

En la Figura 5.44 se puede ver que el número de procesos en post-sincronización se mantiene prácticamente constante entre perfiles (235), actuando como métrica de control del entorno.

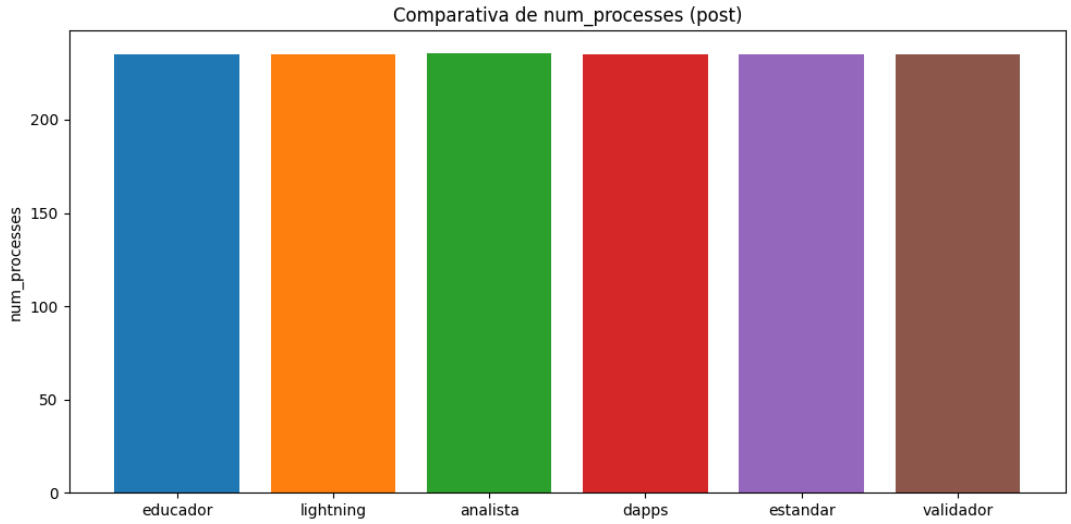

Figura 5.44: Número de procesos activos durante la post-sincronización, comparativa entre perfiles.

Para tener una noción de la evolución temporal, en la Figura 5.45 se muestra el comportamiento del perfil estándar, el cual refleja que las variaciones fueron mínimas durante la medición. El resto de perfiles muestran un patrón similar, sin grandes diferencias.

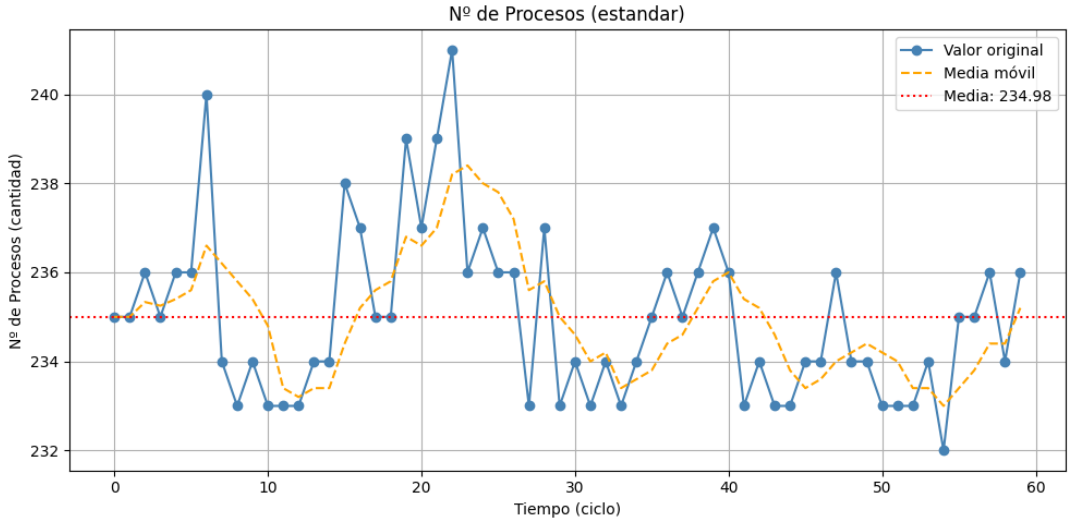

Figura 5.45: Número de procesos en ejecución durante la post-sincronización (perfil Estándar).

Este resultado apoya la afirmación de que no es la ejecución del proceso lo que provoca las diferencias apreciadas en el uso de CPU, memoria, disco y red, sino, por el contrario, el comportamiento del nodo en función del perfil que se ha configurado. Esta métrica, dicho

Métrica	Educador	Lightning	Analista	dApps	Estándar	Validador
CPU (%)	5,71	6,30	6,89	7,97	5,59	4,72
RAM (MB)	1543	1846	1910	1801	1941	1895
Disco (GB)	35,2	35,0	835,8	37,1	823,3	823,7
Red RX (GB, Δ)	0,07	0,08	0,06	0,03	0,07	0,02
Red TX (GB, Δ)	0,01	0,01	0,01	0,01	0,01	0,01
Procesos activos	235	235	235	235	235	235

Tabla 5.5: Valores medios durante la fase de post-sincronización (métricas normalizadas).

Métrica	Educador	Lightning	Analista	dApps	Estándar	Validador
CPU (menor es mejor)						✓
RAM (menor es mejor)	✓					
Disco (menor es mejor)		✓				
Red RX (mayor es mejor)	✓					
Red TX (mayor es mejor)				✓		

Tabla 5.6: Perfiles con mejor rendimiento relativo en post-sincronización

con otras palabras, sirve como un *indicador de control* que permite validar la consistencia del entorno experimental.

Resumen comparativo de la fase de post-sincronización

Los valores medios de las métricas resultantes en post-sincronización obtenidos a partir del fichero `tabla_comparativa.csv` (después de una conversión manual a números legibles) están reflejados en la Tabla 5.5. En la Tabla 5.6 se puede observar de forma visual en qué perfil se obtiene el mejor rendimiento relativo de cada métrica; no se tienen en cuenta procesos porque todos son iguales.

Los resultados actualizados evidencian que, bajo condiciones estables, los perfiles tienden a conservar su patrón de consumo, aunque concretándose mucho más en la red. El perfil **Validador** sigue siendo el que menos CPU utiliza, y el perfil **Educador** el que menos usa la RAM y con un tráfico entrante alto debido a su orientación demostrativa. El perfil **Lightning** permite obtener el mejor equilibrio entre uso de disco y un tráfico bidireccional alto, mientras que el perfil **dApps** destaca por su volumen de tráfico saliente, reflejando su rol activo en entornos de desarrollo. El perfil **Analista** presenta un alto consumo de recursos consistente con su necesidad de mantener índices y filtros activos para realizar el procesamiento de datos. Por último, el perfil **Estándar** tiene un comportamiento

equilibrado en todas las métricas, el cual puede considerarse como la línea base con la que comparar el rendimiento de los otros perfiles.

En estas métricas se puede observar cómo, una vez terminada la sincronización inicial, las diferencias entre perfiles no tienden a desaparecer, sino que se estabilizan en patrones de consumo coherentes con la finalidad funcional de cada perfil.

5.5. Decisiones no triviales y dificultades

El desarrollo del proyecto no solo consistió en implementar instrucciones conocidas, sino más bien consistió en hacer frente a diferentes problemas prácticos y tomar decisiones que condicionaron el resultado final. A continuación se documentan aquellas que han tenido más impacto en la implementación y la fiabilidad de los esquemas experimentales.

Compilación de Bitcoin Core desde fuente

La decisión de optar por compilar **Bitcoin Core v29.0** a partir del código fuente[6], en lugar de instalar binarios precompilados en Ubuntu, es resultado del **control completo** en las configuraciones, como por ejemplo, al habilitar parámetros como `txindex` y `blockfilterindex`, e incluso asegura la integridad del código mismo. El coste fue mayor tiempo de construcción y la necesidad de resolver algunas dependencias, pero fue asumido porque se prefería la **transparencia y reproducibilidad** de los experimentos.

Formato de los ficheros CSV

Al llevar a cabo las primeras ejecuciones durante las primeras semanas, se detectó un error al analizar los logs con `pandas`, pues los valores flotantes generaban columnas adicionales al hacer uso de la coma como separador por defecto. De este modo, los datasets resultaban ilegibles. Como solución, se optó por establecer el valor `";"` como separador en los csv, tal y como se muestra en la cabecera siguiente:

```
ciclo;timestamp;profile;blocks;headers;verificationprogress;
cpu_percent;mem_used_MB;disk_used_MB;net_rx_kB;net_tx_kB;
num_processes;uptime_minutes
```

Si bien esta decisión puede parecer trivial, fue crítica a la hora de asegurar la **integridad de los datos**, pues fue algo que se detectó después de haber realizado las pruebas en los seis perfiles, así que claramente generó problemas posteriores en el análisis, pero que, una vez solucionado, permitió realizar análisis correctos (véase Sección 5.4).

[6]Se utilizó `gcc 13.2.0` y las dependencias recomendadas en la documentación oficial de Bitcoin Core v29.0 (https://github.com/bitcoin/bitcoin).

Integración de la monitorización

Al comienzo se usó un terminal en paralelo para la monitorización de métricas de CPU, RAM, disco y red, pero era una solución que generaba problemas de gestión y errores. Por eso se decidió **integrar la monitorización en `script_tfm.sh`**, mediante el parámetro opcional `-log`, de modo que el script se encargara de que el nodo fuera instalado, configurado y al mismo tiempo monitorizado con el objetivo de alcanzar procesos **controlados y reproducibles** en una única instrucción:

```
./script_tfm.sh -log
```

Definición de perfiles

La configuración de los perfiles de usuario (Validador, Analista, dApps, Lightning, Educador y Estándar) no se llevó a cabo arbitrariamente, sino que estuvo basada en las directrices del tutor del TFM. La dificultad estaba en implementar estas opciones de forma homogénea en el script, para asegurar que cada perfil ejecutara correctamente los parámetros implicados (`prune`, `txindex`, `blockfilterindex`, etc.) y que los resultados pudieran compararse sin sesgos experimentales.

Descartar virtualización y herramientas externas

Se analizó el uso de tecnologías como Docker, máquinas virtuales o plataformas de monitorización. Pero se optó por eliminarlas por dos razones fundamentales:

1. Incorpora una **capa de abstracción** que puede distorsionar las métricas reales de hardware (CPU, RAM, disco).
2. Supone un incremento considerable de la **complejidad de instalación**, lo cual contradice el objetivo que se persigue de ofrecer una herramienta sencilla y de fácil replicación para entornos académicos.

Caracterización del post-sincronización

Otra decisión que se incluyó dentro de la metodología fue la de extender la monitorización durante una hora después de haber efectuado la finalización de la sincronización inicial (*Initial Block Download*). Todo ello, con el propósito de poder captar métricas en una **fase estable**, para poder diferenciar la fase intensiva que estaba siendo ejecutada mediante la IBD de la fase normal de funcionamiento de un nodo. Esta nueva decisión también permitió que se pudieran comparar los perfiles, no sólo en la fase de carga inicial, sino también a partir del momento en que el nodo estaba funcionando normalmente (véase Sección 5.4).

Gestión de permisos y usuario dedicado

Con el objetivo de incrementar la seguridad y emular adecuadamente los entornos de producción, finalmente se tomó la decisión de crear un usuario específico `bitcoin` que se encargara de realizar la ejecución del servicio, y no operar con privilegios de `root`. Esta situación supuso configurar permisos adicionales en directorios y servicios de forma precisa, el cual dio como resultado una mayor robustez y sigue las buenas prácticas de administración de sistemas.

Visualización de métricas

También se introdujo un paso más que consistió en la representación gráfica de los datos utilizando Python. A esta visualización la acompañó una media móvil con objeto de suavizar las series temporales y hacer ver las tendencias generales. Esta media móvil se calculó con la siguiente instrucción en el script `graficos.py`:

```
df["cpu_percent_ma"] = df["cpu_percent"].rolling(window=30, min_periods=1).mean()
```

Toma de decisiones

Si bien cada decisión estaba guiada por un problema particular, todas tenían como meta final la validez y reproducibilidad de los resultados experimentales. La Tabla 5.7 presenta de forma concisa cada decisión, su razón de ser y el efecto en el desarrollo del proyecto.

5.6. Síntesis

El desarrollo expuesto en el presente capítulo puso de manifiesto que el trabajo no solo consistió en realizar pruebas técnicas, sino que también hubo una articulación consistente de diversos componentes: **automatización del despliegue mediante `script_tfm.sh`**, la **definición de perfiles de usuario**, así como la **monitorización ligera en CSV**. Esta combinación permitió realizar un flujo de trabajo reproducible, portable y accesible, y por lo tanto se cumplen los objetivos de conseguir unas métricas exportables bajo condiciones controladas, a la vez que el coste de complejidad es controlado, frente a alternativas más pesadas como contenedores o plataformas externas.

Los resultados de la sincronización y post-sincronización mostraron los patrones esperados de consumo de CPU, RAM, disco y red, y que se explican por las configuraciones de cada perfil. El proceso no fue trivial porque se presentaron problemas: por ejemplo, las depuraciones de formato de los CSV, el compilar desde la fuente, la necesidad de unir la monitorización al script del proyecto, etc. Estas decisiones metodológicas constituyen un **aprendizaje práctico** que confirma la validez de los experimentos y sienta una base para futuras replicaciones.

Decisión	**Motivo**	**Impacto**
Compilación desde fuente	Control total de configuración	Reproducibilidad y transparencia, habilitación de índices avanzados
Separador en CSV (;)	Evitar columnas ilegibles	Consistencia de datos, análisis automatizado fiable
Integrar monitorización en el script	Reducir errores y gestión paralela	Flujo de experimentación simple y replicable
Definir perfiles de usuario	Comparar escenarios reales	Métricas comparables sin sesgos de configuración
Evitar Docker/VMs	Reducir distorsión y complejidad	Métricas fieles al hardware real
Extender monitorización post-sync	Diferenciar IBD de régimen estable	Comparación equilibrada de perfiles
Usuario dedicado `bitcoin`	Buenas prácticas de seguridad	Robustez y realismo en entornos de producción
Media móvil en visualización	Suavizar tendencias en gráficas	Patrones más claros en la interpretación de métricas

Tabla 5.7: Síntesis de decisiones metodológicas y su impacto

En resumen, este capítulo pone de manifiesto que la combinación de **automatización, perfiles diferenciados y análisis comparativo de métricas** ha demostrado ser una metodología eficaz[7] para estudiar nodos Bitcoin en distintas situaciones. La experiencia adquirida confirma que las decisiones tomadas durante la implementación no solo han permitido conseguir los logros propuestos, sino que también han permitido generar un espacio experimental que tiene tanto valor aplicado como académico, que servirá para los trabajos relacionados y las conclusiones finales del TFM.

[7]Este enfoque podría aplicarse también al estudio de otras redes descentralizadas, como *Monero* o *Ethereum*, ajustando las métricas de validación.

6: Trabajos relacionados

6.1. Introducción

El rendimiento de nodos Bitcoin y la evaluación del rendimiento de configuraciones en procesos blockchain ha sido objeto de numerosos estudios en los años recientes. Los estudios incluyen desde trabajos de orientación sistemática y teórica hasta trabajos empíricos y propuestas de simulación. En este apartado se presenta una revisión de los trabajos más relevantes, y la agrupamos en cuatro bloques: (i) trabajos de evaluación de rendimiento en blockchain, (ii) trabajos sobre topologías de red y Lightning Network, (iii) la literatura en lengua hispana y (iv) trabajos de fin de máster en relación con la monitorización de nodos. La comparación con estas aportaciones nos permite situar este TFM en el estado del arte y dar cuenta de su originalidad.

Como complemento divulgativo para aquellos lectores que no tienen formación previa, puede consultarse el libro *Bitcoin For Dummies* que resume de forma simple nociones como *wallets*, direcciones, confirmaciones y comisiones [17].

6.2. Evaluación del rendimiento en blockchain

La evaluación del rendimiento en blockchain es un campo amplio en el que se han generado metodologías muy diversas. Fan et al. [11] ofrecen una revisión sistemática de las técnicas de evaluación existentes, las cuales parecen clasificarse, de acuerdo con los autores, en cuatro familias: *benchmarking*, simulación, modelado analítico y experimentación empírica. Este mismo trabajo pone de manifiesto la necesidad de contar con métricas que sean comparables y con herramientas que sean reproducibles, lo que se relaciona de forma directa con la filosofía del presente TFM.

Wang et al. [22], en cambio, analizan la red Bitcoin desde el punto de vista de los pools de minería[1], analizando, por un lado, la interconectabilidad, los delays de la propagación

[1]Un pool de minería son un grupo de mineros que combinan su poder computacional para aumentar las probabilidades de encontrar un bloque y, de este modo, lograr recibir una recompensa de forma más frecuente.

en la red y sus métricas de seguridad. Si bien su enfoque está centrado en un uso de la capa de red y en el comportamiento colectivo de los mineros, ofrece un marco útil para comparar el consumo de recursos en los nodos.

Alsahan, Lasla y Abdallah [3], por otro lado, presentan una propuesta de un simulador de red local basado en la virtualización ligera, cuyo objetivo es la recreación de topologías de nodos en entornos controlados para medir el rendimiento sin la necesidad de utilizar la red pública. Distinto a este tipo de propuestas, el TFM que aquí se presenta opta por realizar una evaluación sobre hardware real y en condiciones de máxima sincronización, lo que permite obtener resultados directamente aplicables.

6.3. Estudios sobre Lightning Network y topologías

La cuestión del análisis de redes de segunda capa ha sido también objeto de la atención de algunos otros estudios. En este sentido, Zabka et al. [25], realizan una evaluación empírica de los nodos y canales de la Lightning Network, gracias a una clasificación de las implementaciones y el estudio de su rendimiento. Si bien el TFM no aborda Lightning de forma directa, sí que incluye un perfil concreto para este entorno, lo que le conecta con las conclusiones de este trabajo.

Desde una perspectiva más teórica, Wright [23] revisa la redundancia de los nodos completos de Bitcoin y la función que los mismos obtienen en la topología de la red. La conclusión de tal trabajo es que los nodos no mineros, a pesar de ser redundantes en el proceso de validación, cumplen un rol crucial en la resiliencia de la red. Este tipo de aportes justifican los perfiles como el de validador, que orientan hacia la máxima soberanía y la verificación independiente de la red.

Finalmente, Riquet et al. [19] examinan el efecto que tienen las distintas topologías de red sobre el rendimiento del blockchain usando el framework Lilith[2]; el trabajo muestra que esta topología de los nodos puede ser un factor a tener en cuenta en la latencia y el throughput de la red. El enfoque de este TFM, aunque no contempla la topología de la red, añade un espacio complementario revisando el consumo de recursos internos (CPU, RAM, disco, red) en las distintas configuraciones.

Para una visión introductoria y no técnica sobre el uso de Bitcoin por parte del usuario (gestión de *wallets*, direcciones, confirmaciones y nociones de seguridad) la guía divulgativa *Bitcoin For Dummies* resulta útil [17].

[2]Lilith es un entorno creado para el artículo académico de la Université de Toulouse, para simular y evaluar el rendimiento de redes blockchain bajo distintas topologías de conexión entre nodos. Permite estudiar cómo la estructura de la red afecta la latencia y la capacidad de procesamiento. No debe confundirse con otros frameworks llamados “Lilith” relacionados con computación paralela que suelen aparecer en buscadores.

6.4. Aportaciones en español

La literatura en lengua española también ha dado lugar a estudios importantes en el mapeo de Bitcoin y la tecnología blockchain. Acuña [1] presenta una revisión general sobre Bitcoin y su repercusión en el sistema financiero, con un marco conceptual que permite asimilar las diferencias entre los sistemas de banca tradicionales y aquellas redes descentralizadas. Más recientemente, Álvarez Pizarro, Sánchez Galvis y Zabala Vargas [26] realizan un estudio comparativo para la selección de una red blockchain pública en el contexto de notificaciones certificadas, en donde la escalabilidad, seguridad, confianza y costos son objetos tenidos en cuenta, ofreciendo un enfoque aplicado, cercano a la realidad de distintas instituciones en búsqueda de soluciones basadas en blockchain. En este sentido, aunque no es el núcleo del estudio, la investigación enfatiza los beneficios de evaluar configuraciones y métricas en contextos concretos, un elemento con el que coincide la motivación del presente TFM. En el ámbito de la minería, Medina [15] discute los diferentes niveles de rendimiento en entornos de *merged mining*[3], estudiando la eficiencia de los recursos bajo esta perspectiva. Aunque también este estudio se centra más en la minería que en los nodos completos, se puede decir que sus conclusiones sirven para apoyar la necesidad de mediciones empíricas de rendimiento en entornos reales.

6.5. Trabajos de fin de máster relacionados

Aparte de la literatura científica presentada en revistas académicas o congresos, hay trabajos de fin de máster que están ya relacionados con el tema de este proyecto. Pérez Conte [18] expone un trabajo de fin de máster en el que se ocupa de la identificación de nodos mineros mediante scripts que permiten monitorizar y analizar la red. Aunque la presencia del trabajo es mucho más sencilla y se centra casi exclusivamente en la detección de mineros, su parte práctica y basada en scripts es eminentemente coincidente con la filosofía del presente trabajo.

En la misma línea, otros trabajos de máster han presentado el despliegue de nodos y la monitorización de métricas concretas. Este trabajo de fin de máster amplía este campo, ya que combina **automatización de instalación**, **definición de perfiles de usuario** y **análisis comparativo de recursos**, lo que constituye una aportación más integral en esta área.

6.6. Síntesis

Los trabajos examinados presentan un panorama diverso: revisiones sistemáticas [11], simulaciones en entornos controlados [3], evaluaciones de topologías de red [19], y estudios empíricos de la Lightning Network [25]. A ellos se les añaden contribuciones en español sobre fundamentos [1], sobre aplicaciones institucionales [26] y minado combinado [15].

[3]Merged mining: Designa el proceso mediante el cual un mismo minero puede participar simultáneamente en la minería de dos o más criptomonedas diferentes utilizando la misma potencia de cálculo.

Por último, los trabajos de fin de máster [18] presentan antecedentes directos en la automatización del uso de scripts en la automatización de la experimentación.

En su conjunto, la literatura existente respalda la validez del tema, pero al mismo tiempo muestra una **carencia de estudios empíricos sobre el consumo de recursos de hardware en nodos completos de Bitcoin**. La mayoría de las publicaciones se focalizan en la capa de red, en simulaciones y en aspectos teóricos. En este sentido, la aportación original de este TFM es la integración de la monitorización ligera de CPU, RAM, disco y red, en la automatización del despliegue para diferentes perfiles de usuario, con datos reproducibles y comparables complementarios a las aproximaciones anteriores.

7: Conclusiones y Líneas de trabajo futuras

Las aportaciones principales de este TFM son las siguientes:

- Se ha realizado un **script en Bash**, denominado `"script_tfm.sh"`, para la instalación automática y el arranque de nodos Bitcoin Core en un sistema operativo GNU/Linux.
- Se han definido **seis perfiles de usuario**: Validador, Analista, dApps, Lightning, Educador y Estándar, los cuales representan diferentes contextos operativos y niveles de uso.
- Se ha incorporado un sistema de **monitorización de métricas** (CPU, RAM, disco y red) que se registró en un CSV durante las fases de sincronización y post-sincronización. Todo esto mediante el flag "log".
- Se ha desarrollado un **módulo de análisis en Python** (`graficos.py`) a partir de los resultados experimentales (CSV), el cual tiene como objetivo procesar y representar visualmente estos resultados experimentales de manera gráfica, para poder realizar los estudios comparativos del TFM.
- Se ha demostrado la **importancia de los perfiles funcionales** definidos para el TFM como método de comparación de rendimiento y consumo de recursos.
- Se ha creado una **metodología reproducible y portable** para evaluar configuraciones de nodos blockchain, con potencial de extensión a otras redes (Monero, Ethereum, etc.).
- Se obtiene como aporte una **herramienta libre y educativa** con valor didáctico y de investigación para la enseñanza universitaria y para la divulgación tecnológica.

Este Trabajo Fin de Máster ha conseguido evidenciar la posibilidad de poder automatizar de forma integral el empaquetado como despliegue y la monitorización de nodos

Bitcoin Core en entornos Linux con diferentes perfiles de usuario (Validador, Analista, dApps, Lightning, Educador y Estándar), además de facilitar datos empíricos reproducibles sobre el registro de recursos en las fases de sincronización y post-sincronización. El trabajo no consistió solo en implementar procedimientos que ya se conocían, sino que requirió tomar decisiones técnicas que condicionaron el resultado final y aportaron un carácter metodológico propio al proyecto.

Los resultados experimentales durante la fase inicial de sincronización (IBD) evidenciaron cómo la CPU y el disco son los recursos más limitantes en los perfiles completos (Validador y Analista), mientras que todos los perfiles ligeros (Educador, dApps, Lightning) transfieren la carga hacia la red y la CPU. El perfil Educador fue el que utilizó muy poco espacio en disco y memoria RAM, aunque con uso intensivo de CPU y tráfico de red, lo que lo transforma en una buena opción para entornos didácticos, pero sorprendentemente malo en entornos del hardware con conectividad limitada; el perfil dApps, en cambio, es el perfil con un verdadero equilibrio en el uso de CPU y almacenamiento, por lo que se transforma en útil para desarrolladores cuando es necesario ligereza sin renunciar al uso avanzado de funciones de consulta. En la fase de post-sincronización, las métricas se estabilizan y las diferencias extremas de los perfiles disminuyen, confirmando que la fase más exigente para la infraestructura es, efectivamente, la inicial.

Desde la perspectiva técnica, la compilación de Bitcoin Core a partir del código fuente propició transparencia y control total de las opciones de configuración, requiriendo, sin embargo, poca resolución de dependencias. La monitorización integrada al propio script `script_tfm.sh` con logs en CSV procesados en Python resultó una alternativa ligera y segura en comparación con alternativas más complejas sin perder rigor en los resultados. La definición de perfiles automáticos en el fichero `bitcoin.conf` supuso, además, estandarizar las comparativas y reproducir escenarios de uso real homogéneamente, lo que supuso valor académico y práctico. La modularidad en Bash y Python supuso un flujo de trabajo reproducible y portable alineado con los objetivos del TFM.

Si bien se han cumplido los objetivos inicialmente planteados, el trabajo también ha puesto de manifiesto algunas limitaciones. En primer lugar, la experimentación se ha llevado a cabo en un único entorno de hardware, lo que impide generalizar los resultados obtenidos. En segundo lugar, las métricas utilizadas únicamente se centraron en el uso de CPU, RAM, disco y red, quedando fuera de la evaluación otros aspectos no menos importantes, como la latencia de propagación de bloques o el consumo energético. En tercer lugar, el análisis de seguridad, resiliencia y topología de red quedó fuera del alcance que se había definido para este trabajo, aunque es cierto que es una línea natural a considerar en futuras extensiones.

Por tal razón, el presente trabajo tiene múltiples caminos de investigación y mejora. Por un lado, replicar los experimentos en hardware heterogéneo, desde dispositivos de bajo consumo como Raspberry Pi a servidores de alto rendimiento, permite tener una visión de aspecto mucho más grande sobre la viabilidad de cada perfil. La inclusión de nuevas métricas, como por ejemplo latencia o eficiencia energética, permite tener un marco de evaluación más exhaustivo. El análisis del impacto de la topología de red, así como la

evaluación de redes de segunda capa como Lightning Network[1], constituye otra línea de gran interés. Y, por último, dicha integración del sistema en herramientas de monitoreo más avanzadas permitiría ampliar las capacidades de visualización y de alerta en los ámbitos productivos. Finalmente, el perfil Educador puede llevarse a cabo en un kit educativo acompañado de manuales y de simuladores, siendo así un medio para la difusión de la tecnología blockchain en el ámbito universitario.

Para finalizar, este capítulo asegura que la combinación de automatización, perfiles y comparativa de métricas es un método adecuado para estudiar nodos bitcoin en distintos contextos. Las decisiones metodológicas durante el desarrollo no solo permitieron cumplir los objetivos académicos del TFM, sino que, a su vez, dejaron una buena base para futuros estudios y una herramienta práctica con valor aplicado y replicable.

[1]Lightning Network es una red de segunda capa construida sobre la blockchain de Bitcoin diseñada para realizar micropagos instantáneos y de bajo coste entre usuarios, evitando la necesidad de registrar cada micropago en la blockchain. Lightning Network funciona mediante canales de pago entre nodos que sólo se liquidan en la blockchain cuando se abren o se cierran, mejorando la escalabilidad y la velocidad del sistema.

Apéndices

Apéndice A

Plan de Proyecto

A.1. Introducción

El presente apéndice figura el plan de proyecto que se seguira para la realización del Trabajo Fin de Máster (TFM). El desarrollo del mismo deviene de la planificación, ejecución y control de las tareas que giran en torno a la automatización de la instalación del *Bitcoin Core*, la definición de los perfiles de usuario, la monitorización del rendimiento del nodo y el análisis de resultados, el cual fue comparativo. La gestión del proyecto ha sido desarrollada siguiendo los lineamientos del PMBOK adaptados al contexto de un máster. A continuación, se definen los principales hitos junto con la planificación temporal y el estudio de viabilidad en sus dimensiones técnica, económica y legal.

A.2. Planificación temporal

La planificación está estructurada en torno a los hitos relevantes del proyecto[1], desde el inicio oficial hasta el de las pruebas experimentales de óptima calidad y de la redacción del informe correspondiente. En la tabla A.1 se resumen los hitos y los entregables más importantes.

A.3. Estudio de viabilidad

El análisis de viabilidad analiza la factibilidad del proyecto desde diferentes puntos de vista: económico, legal y técnico. Además de incluir los riesgos que se hayan podido identificar, así como las correspondientes acciones para mitigarlos.

[1]El desarrollo de cada hito fue documentado en el capítulo 5, siguiendo la secuencia aquí representada.

Hito	Fecha	Entregable
H1: Inicio oficial del proyecto	06/06/2025	Planificación inicial y definición del alcance
H2: Inicio del desarrollo del script	13/06/2025	Primera versión de `script_tfm.sh`
H3: Incidencia externa (apagón)	01/07/2025	Replanificación y ajuste de cronograma
H4: Ejecución de pruebas de sincronización	08–14/07/2025	CSV de métricas de IBD por perfil
H5: Finalización de pruebas	21/07/2025	CSV post-sync y datos comparativos
H6: Redacción de la memoria TFM	22/07/2025 – 30/09/2025	Informe preliminar y apéndices
H7: Versión final del TFM	01/10/2025	Documento final entregable

Tabla A.1: Hitos y entregables del proyecto

Viabilidad económica

El proyecto se ha desarrollado en el marco académico del Máster, es decir, se han utilizando exclusivamente recursos disponibles de manera gratuita o bien institucional:

- **Hardware**: Servidor virtual de la Universidad de Valladolid (sin coste adicional)
- **Software**: Sistema GNU/Linux, compiladores y librerías de código abierto (`gcc`, `cmake`, `git`, etc.).
- **Bitcoin Core**: Proyecto libre, bajo licencia MIT.
- **Lenguajes de programación**: Bash y Python (ambos en entornos que ya ven la GNU/Linux).
- **Análisis y visualización**: Librerías Python (`pandas`, `matplotlib`) que se presentan como software libre.

Si bien no se incurrió en gastos monetarios directos, el desarrollo sí supuso un consumo real de recursos en términos de tiempo, que consideramos importante. Conforme al diagrama de hitos que se estableció en la tabla A.1, se estima un total cifrado de **150–180 horas de trabajo técnico y documentación**, el cual habrá demandado un esfuerzo activo, distribuido a partes iguales entre la planificación, el desarrollo de script principal, la realización de pruebas de sincronización, el análisis de métricas y la redacción del TFM.

Si se considera el salario medio de un ingeniero informático en España, el cual llega a ser **13,85 €/hora** según *Talent.com* (2025), el coste teórico asociado al esfuerzo total puede situarse entre **2.000 €** y **2.500 €**, aunque este valor no representa un gasto real, sí puede permitir dimensionar el esfuerzo y los recursos dedicados desde la gestión de costes, reflejando un uso eficiente de los medios institucionales y del software libre. Por eso, el proyecto tiene una alta viabilidad económica no solo porque, al no requerir mayor inversión para licencias o infraestructura, nos ofrece un importante retorno en términos académicos y tecnológicos en relación con los medios empleados.

Viabilidad legal

Se ha revisado el marco legal y de permisos para la utilización de los productos generados y evaluados en el TFM:

- **Bitcoin Core** se encuentra publicado bajo licencia MIT, que permite su uso, modificación y distribución sin restricciones de ningún tipo.
- Por otro lado, el **script `script_tfm.sh`** así como el **código `graficos.py`** desarrollados en el contexto de este TFM, están pensados para luego ser públicos y bajo licencia MIT.
- Finalmente, los datos experimentales generados y utilizados en el TFM (`CSV`, logs y gráficas) se consideran de dominio académico y se entregan como material complementario al informe, con la voluntad de ofrecer su uso sin restricciones de cualquier tipo de copyright, etc.

Por lo tanto, no existen conflictos legales que impidan el desarrollo, publicación y difusión de los resultados del proyecto.

Viabilidad técnica

Desde un punto de vista técnico, la viabilidad ha estado condicionada por:

- **Compatibilidad**: El script es ejecutable en cualquier distribución de GNU/Linux que tenga soporte de `systemd` y gestor de paquetes APT.
- **Requisitos mínimos**[2]: Al menos 2 cores de CPU, 4 GB de RAM y 40 GB de disco para perfiles podados; para perfiles completos se recomienda disponer de 1 TB de disco y 8 GB de RAM.
- **Escalabilidad**: Los perfiles Estándar, Validador, Analista, dApps, Lightning y Educador ofrecen un rango de perfiles suficientes adaptados a los diferentes escenarios de usuario.

[2] A partir de https://bitcoin-org.translate.goog/en/bitcoin-core/features/requirements, pero adaptados a la realidad vista en el proyecto del TFM.

- **Robustez**: El uso de `systemd` permite tener la seguridad de que el servicio `bitcoind` correrá de forma persistente y que pueda recuperarse sin problemas ante errores o fallos.

Riesgos identificados

Conforme a la gestión de los proyectos con base en el PMBOK, se encontraron los siguientes riesgos más significativos:

- **Fallo en la infraestructura de pruebas**: El apagón eléctrico dentro de la universidad el 01/07/2025[3], hizo que fuera necesario volver a programar actividades críticas. Mitigación: recuperación periódica de los datos y planificación flexible.
- **Sobrecarga de recursos**: En equipos con menos de 4 GB de RAM o con el espacio en disco insuficiente, los perfiles completos pueden ser inviables. Mitigación: uso de perfiles podados (dApps, Lightning, Educador).
- **Retrasos en la sincronización**: La descarga inicial (IBD) puede tardar varios días. Mitigación: pruebas distribuidas en paralelo y limitación de los perfiles a condiciones realistas.

Para concluir, se consideró que el proyecto es viable en todo su alcance, cumpliendo con las restricciones académicas, legales y técnicas propuestas.

[3]Ese dia no se pudo realizar la conexión por medio de SSH al servidor, por lo cual se le informó al tutor, quien respondió indicando que se generó este problema dentro de la universidad.

Apéndice B

Especificación de Requisitos

B.1. Introducción

El presente apéndice, tal y como se ha realizado, establece de manera verificable los requerimientos del sistema desarrollado en el TFM: un script de automatización para la instalación, configuración y monitorización de *Bitcoin Core* en GNU/Linux, adaptado para distintos perfiles de usuario; y un módulo de análisis que gestiona las métricas generadas. El propósito es que el lector pueda conocer *qué* debe presentar el sistema (requerimientos funcionales) y *cómo* debe comportarse (requerimientos no funcionales), así como su alcance, la interfaz y los criterios de aceptación. Los requerimientos definidos aquí han sido verificados experimentalmente en la sección 5.4. Por último, la sección sigue una ligera estructura de SRS (IEEE 830/29148)[1].

Alcance

El sistema contempla: (i) la instalación y configuración automática de *Bitcoin Core* v29.0; (ii) la selección y aplicación de perfiles de ejecución (valores en `bitcoin.conf`; (iii) la implementación como servicio `systemd`; (iv) el registro ligero de métricas en CSV durante IBD más 60 minutos post-sincronización; (v) la creación de tablas y figuras a partir de los CSV. Fuera del alcance de este sistema se encuentra la seguridad avanzada (hardening de kernel, firewalls), alta disponibilidad, interactividad de GUI, monitorización externa, contenedores, y monitorización de la topología de red.

Definiciones y abreviaturas

- **IBD**: *Initial Block Download*, sincronización inicial de la cadena.

[1]Las normas `IEEE 830` e `IEEE 29148` son estándares internacionales en ingeniería de software desarrollados por el Institute of Electrical and Electronics Engineers (IEEE). En ambas normas se establecen las distintas pautas de redacción y de estructura para la elaboración de los documentos de especificación de requisitos del software (SRS, Software Requirements Specification).

- **CSV**: Ficheros con separador `";"` (punto y coma) para compatibilidad regional.
- **Perfil**: Conjunto de parámetros efectivos de `bitcoin.conf` aplicados por el script.

B.2. Objetivos generales

1. **O1** — Automatizar, completamente, el despliegue de un nodo *Bitcoin Core* en GNU/Linux.
2. **O2** — Permitir la configuración del nodo mediante la utilización de *perfiles* pensados para casos de uso reales.
3. **O3** — Registrar o almacenar métricas de sistema y de sincronización (IBD y post-sync) en CSV que sean reutilizables.
4. **O4** — Exponer análisis reproducibles (tablas/figuras) a partir desde los CSV generados.
5. **O5** — Garantizar reproducibilidad y simplicidad (sin virtualización; dependencias mínimas).

B.3. Catálogo de requisitos

A continuación se listan requisitos funcionales (RF) y no funcionales (RNF). Prioridad: **M** (Must), **S** (Should), **C** (Could).

Requisitos funcionales (RF)

RF-01 (M) Instalación asistida El sistema *debe* verificar el entorno (OS, kernel, CPU, RAM, disco), además de instalar dependencias necesarias vía `apt`, para luego compilar *Bitcoin Core* v29.0 desde fuente y crear el usuario `bitcoin` junto al `datadir` en `/opt/BLOCKCHAIN`.

RF-02 (M) Configuración de servicio El sistema *debe* crear y habilitar `bitcoind.service` en `systemd` con `User=bitcoin`, `Restart=on-failure` y `PIDFile` gestionado.

RF-03 (M) Generación de credenciales RPC El sistema *debe* generar `rpcauth` y escribir `/etc/bitcoin/bitcoin.conf` y `/home/bitcoin/.bitcoin/bitcoin.conf` con `datadir=/opt/BLOCKCHAIN`.

RF-04 (M) Selección y aplicación de perfiles El sistema *debe* permitir seleccionar un perfil y escribir parámetros efectivos en `/etc/bitcoin/bitcoin.conf`. Considerando estos perfiles y parámetros:

- **Estándar**: `txindex=1`[2].
- **Validador**: `prune=0`, `txindex=1`.
- **Analista**: `prune=0`, `txindex=1`, `blockfilterindex=1`.
- **dApps**: `prune=550`, `txindex=0`, `blockfilterindex=0`.
- **Lightning**: `prune=1000`, `txindex=0`, `blockfilterindex=0`, `mempoolexpiry=336`, `maxconnections=40`.
- **Educador**: `prune=1000`, `txindex=0`, `blockfilterindex=0`, `assumevalid=0`, `maxconnections=20`, `mempoolexpiry=72`, `dbcache=100`.

RF-05 (M) Detección de IBD El sistema *debe* detectar el estado `initialblockdownload` vía `bitcoin-cli getblockchaininfo` para poder identificar si ya está sincronizando.

RF-06 (M) Registro de métricas (modo `-log`) El sistema *debe* registrar durante IBD, muestreando cada 60 s, en `/var/log/bitcoin-monitor/` un fichero `sync_log_{perfil}_{timestamp}.csv` con cabecera:

```
ciclo;timestamp;profile;blocks;headers;verificationprogress;
cpu_percent;mem_percent;disk_used_MB;net_rx_kB;net_tx_kB
```

Los contadores de red (net_rx_kB, net_tx_kB) son acumulativos del sistema (desde el arranque) tal como reporta /sys/class/net/*/statistics/*_bytes.

RF-07 (M) Registro post-sincronización Tras finalizar IBD, el sistema *debe* registrar 60 minutos adicionales (cada 60 s) en `post_sync_metrics_{perfil}_{timestamp}-.csv` con cabecera:

```
ciclo;timestamp;profile;cpu_percent;cpu_cores;load_1min;load_5min;
load_15min;mem_used_MB;mem_free_MB;swap_used_MB;swap_free_MB;
disk_used_MB;disk_avail_MB;net_rx_kB;net_tx_kB;uptime_minutes;
num_processes
```

Los valores en post-sync se interpretan como serie temporal muestreada, útil para derivar deltas por intervalo de 60 s.

RF-08 (S) Cambio de perfil en sistema ya instalado El sistema *debería* permitir reconfigurar el perfil, reescribir `bitcoin.conf` y reiniciar el servicio.

[2]Se adopta `txindex=1` como línea base experimental para permitir análisis homogéneo y consultas sobre el histórico completo sin reindexación posterior. Aunque el valor por defecto de Bitcoin Core es `txindex=0`, esta decisión se justificó en el Cap. 5.2 y se implementa en `script\_tfm.sh`.

RF-09 (S) Desinstalación El sistema *debería* ofrecer desinstalación segura: parar servicio, deshabilitar, retirar binarios, configs y `/opt/BLOCKCHAIN` bajo confirmación expresa.

RF-10 (S) Análisis de CSV Debe existir un script `graficos.py` que lea los CSV (con separador `;`), genere tablas (`resumen_completo.csv`, `resumen_estadistico.csv`, `tabla_comparativa.csv`) y figuras en `img/metrics/` para posterior análisis.

Requisitos no funcionales (RNF)

RNF-01 (M) Plataforma GNU/Linux x86_64 con `systemd` y privilegios `root/sudo`.

RNF-02 (M) Reproducibilidad Versión fijada *Bitcoin Core* v29.0; perfiles deterministas.

RNF-03 (M) Simplicidad Sin contenedores ni orquestadores[3]; dependencias por `apt`.

RNF-04 (M) Seguridad básica Servicio bajo usuario `bitcoin` (no `root`); `rpcauth` generado; permisos `600` en `~/bitcoin/bitcoin.conf`.

RNF-05 (M) Datos CSV con separador `;` y codificación UTF-8; frecuencia de muestreo 60 s.

RNF-06 (S) Portabilidad Validado en Ubuntu 24.04 (kernel 6.8); *debería* funcionar en derivadas con `systemd`.

RNF-07 (S) Robustez Gestión de errores y mensajes claros en instalación y servicio.

RNF-08 (C) Rendimiento Compilación con soporte opcional ZMQ[4] seleccionable.

Interfaz externas (EIR)

- **CLI principal**: `sudo ./script_tfm.sh [ -log ]`.
- **Servicio**: `systemctl start|stop|status bitcoind`, `journalctl -u bitcoind -f`.
- **Bitcoin CLI**: `bitcoin-cli getblockchaininfo`.
- **Ficheros clave**:

[3]Orquestador: Software que coordina y automatiza el despliegue, la configuración, la ejecución y el escalado de múltiples contenedores o servicios dentro de un sistema distribuido.

[4]ZeroMQ (ZMQ) es una librería de mensajería rápida que permite a Bitcoin Core publicar de forma instantánea la llegada de nuevos bloques o transacciones para otros programas o servicios que estén conectados.

```
/etc/bitcoin/bitcoin.conf
/home/bitcoin/.bitcoin/bitcoin.conf
/opt/BLOCKCHAIN
/var/log/bitcoin-monitor/*.csv
```

B.4. Especificación de requisitos

Historias de usuario (resumen)

HU-1 - Instalación completa. Como *administrador Linux*, quiero ejecutar un solo comando que se encargue de configurar y activar un nodo `bitcoind` como parte de un servicio con el fin de evitar errores manuales. *Criterios de aceptación*: Servicio dede estar `active (running)`, `rpcauth` debe estar generado y `datadir` debe estar creado con permisos adecuados.

HU-2 - Selección de perfil. Como *usuario técnico*, deseo seleccionar un perfil ya sea *Validador, Analista, dApps, Lightning, Educador o Estándar*, que adapte el fichero `bitcoin.conf` a mi caso de uso. *Criterios de aceptación*: Que los parámetros sean efectivamente escritos, que el servicio se reinicie y que el `getblockchaininfo` funcione.

HU-3 - Registro de métricas. Como *investigador*, tengo la intención de registrar las métricas de CPU, RAM, disco y red durante IBD, y además, 60 minutos de post-sync para comparar perfiles específicos del hardware utilizado. *Criterios de aceptación*: CSV con cabeceras específicas, muestreo de 60 segundos y finalización automática.

HU-4 - Análisis. Como *investigador/docente*, deseo generar tablas y figuras a partir de los CSV para ser utilizadas dentro del TFM. *Criterios de aceptación*: Tener un fichero CSV que pueda ser convertido a gráficos y ser generados dentro de alguna carpeta del equipo.

Casos de uso (UC)

UC-01 Instalar nodo Actor: Administrador. Flujo. Ejecutar script ./script.tfm → comprobar condiciones en equipo para instalar el nodo → instalar dependencias que faltan → compilación de ficheros bitcoin v29.0 → hacer rpcauth y configuraciones adecuadas → crear servicio → crear fichero según perfil → arranque bitcoin. Postcondición: `bitcoind` activo.

UC-02 Cambiar perfil Actor: Administrador. Flujo: menú perfiles → seleccionar perfil → reescritura bitcoin.conf → `systemctl restart`. Postcondición: parámetros efectivos.

UC-03 Registrar métricas Actor: Investigador. Flujo: ejecutar con `./script_tmf -log` → generar CSV de IBD → generar CSV de post-sync. Postcondición: dos CSV por ejecución.

UC-04 Analizar CSV Actor: Investigador/Docente. Flujo: `python3 graficos.py` → generar tablas y figuras. Postcondición: Contar con CSV generados `./script_tmf -log`.

Criterios de aceptación (extracto)

- **CA-01** (**RF-01,02,03**): Tras instalación, `systemctl status bitcoind` muestra `active (running)` y `bitcoin-cli getblockchaininfo` devuelve JSON válido.
- **CA-02** (**RF-04**): Al seleccionar cada perfil, `/etc/bitcoin/bitcoin.conf` contiene exactamente los parámetros esperados (sin residuos de configuraciones previas).
- **CA-03** (**RF-06,07**, **RNF-05**): Los CSV existen, tienen las cabeceras definidas y *solo* usan `";"` como separador; existen 60 filas en post-sync (registro de una hora).
- **CA-04** (**RF-10**): `graficos.py` genera, sin errores, las tablas y figuras en las rutas indicadas.

Restricciones y supuestos

- Se requiere `apt` (Debian/Ubuntu o derivadas) y `systemd`.
- Se requiere conectividad estable a Internet para clonado/compilación y sincronización.
- La ruta `/opt/BLOCKCHAIN` debe estar en un volumen con espacio suficiente de acuerdo a los perfiles.

Estructura de entrega (informativa)

```
TFM/
  docs/
    csv/
      resumen_completo.csv
      resumen_estadistico.csv
      tabla_comparativa.csv
      metrics/
        sync/   # CSV de IBD (por perfil, ejemplo)
        post/   # CSV post-sync (por perfil, ejemplo)
    logs/       # logs de instalación/ejecución (entregados)
  img/
    metrics/
```

Objetivo	**Requisitos asociados**
O1	RF-01, RF-02, RF-03, RNF-01, RNF-03
O2	RF-04, RF-08
O3	RF-05, RF-06, RF-07, RNF-05
O4	RF-10, RNF-05
O5	RNF-02, RNF-03, RNF-06, RNF-07

Tabla B.1: Matriz de trazabilidad (O → RF/RNF)

```
    sync/         # figuras por perfil durante IBD (ejemplo)
    post/         # figuras por perfil en post-sync (ejemplo)
    comparativas/ # figuras comparativas (ejemplo)
source/
  script_tfm.sh   # instalación, perfiles, servicio y monitorización
  graficos.py     # análisis de CSV y generación de figuras/tablas
```

Matriz de trazabilidad (O → RF/RNF)

En la Tabla B.1 se puede apreciar la Matriz de trazabilidad.

Notas de verificación — La conformidad de RF/RNF se valida con las pruebas de humo[5], de perfiles y de monitorización descritas en el Apéndice D.5, y con la generación de tablas/figuras reproducibles a partir de los CSV.

[5]Pruebas de humo: Tipo de prueba inicial rápida y básica cuyo objetivo es verificar que el sistema arranca correctamente y que sus funciones esenciales no fallan de inmediato.

Apéndice C

Documento de Diseño

C.1. Introducción

Este apéndice está dedicado a la descripción del diseño de la solución software desarrollada: una herramienta de automatización y medición para nodos *Bitcoin Core* v29.0. El diseño persigue tres objetivos: (1) Facilitar los despliegues reproducibles en GNU/Linux; (2) Permitir los perfiles de uso con configuraciones diferenciadas; y (3) Capturar métricas ligeras y comparables para la sincronización inicial (IBD) y un estado posterior, en modo ya estable (post–sync).

El apéndice se centra en el **qué** y en el **porqué** de la solución (datos, flujos, arquitectura). Por otra parte, los **cómo** (instalación, comandos, estructura de entrega, extensiones) se documentan en la Documentación del Programador; en cambio, el uso final por parte de los usuarios no técnicos en la Documentación de usuario.

C.2. Diseño de datos

Fuentes de datos y formato

Las métricas se obtienen mediante utilidades estándar de GNU/Linux (`top`, `free`, `df`, `uptime`, lecturas de `/proc` y `/sys`) y mediante bitcoin-cli getblockchaininfo para el estado de sincronización. Y el registro de los resultados se realiza en ficheros CSV con separador `";"`, cada **60 segundos**. Se distinguen 2 tipos de ficheros CSV:

- **CSV de IBD (sincronización inicial)**: ficheros `sync_log_{perfil}_{timestamp}.csv`
- **CSV de post-sync (60 minutos)**: ficheros `post_sync_metrics_{perfil}_{timestamp}.csv`

Campo	Tipo	Descripción / Unidad
`ciclo`	entero	contador de muestras (0,1,2,...)
`timestamp`	texto	fecha–hora ISO local de la muestra
`profile`	texto	nombre del perfil activo
`blocks`	entero	altura de bloques del nodo
`headers`	entero	altura de cabeceras conocida
`verificationpro`	real	progreso de verificación [0..1]
`cpu_percent`	real	uso de CPU (%)
`mem_percent`	real	uso de RAM (%)
`disk_used_MB`	entero	espacio usado en `/opt/BLOCKCHAIN` (MB)
`net_rx_kB`	entero	bytes recibidos/`1024` (kB) en interfaz activa
`net_tx_kB`	entero	bytes enviados/`1024` (kB) en interfaz activa

Tabla C.1: Esquema de *sync* (IBD)

Esquema de *sync* (IBD)

Cabe destacar que durante la fase de sincronización inicial (IBD), el `script_tfm.sh` registra la memoria en porcentaje (`mem_percent`); ya entendido esto, se procede a presentar la cabecera y significado de campos en la Tabla C.1.

Esquema de *post-sync* (60 min)

Por su parte, durante la fase de post-sincronización, el `script_tfm.sh` registra la memoria en unidades absolutas (`mem_used_MB`). Por lo cual, sabiendo esto, se procede a presentar la cabecera y significado de campos en la Tabla C.2:

Artefactos derivados

A partir de los CSV, el análisis (`graficos.py`) genera:

- **Tablas**: `resumen_completo.csv`, `resumen_estadistico.csv`, `tabla_comparativa.csv`
- **Figuras**: series temporales y comparativas (carpetas `sync/`, `post/`, `comparativas/`)

Campo	Tipo	Descripción / Unidad
`ciclo`	entero	contador de muestras (0..60)
`timestamp`	texto	fecha–hora ISO local de la muestra
`profile`	texto	nombre del perfil activo
`cpu_percent`	real	uso de CPU (%)
`cpu_cores`	entero	núm. de cores detectados
`load_1min`	real	carga media 1 min
`load_5min`	real	carga media 5 min
`load_15min`	real	carga media 15 min
`mem_used_MB`	entero	RAM usada (MB)
`mem_free_MB`	entero	RAM libre (MB)
`swap_used_MB`	entero	swap usada (MB)
`swap_free_MB`	entero	swap libre (MB)
`disk_used_MB`	entero	usado en `/opt/BLOCKCHAIN` (MB)
`disk_avail_MB`	entero	disponible en `/opt/BLOCKCHAIN` (MB)
`net_rx_kB`	entero	bytes recibidos/`1024` (kB)
`net_tx_kB`	entero	bytes enviados/`1024` (kB)
`uptime_minutes`	entero	minutos de *uptime* del sistema
`num_processes`	entero	procesos activos totales

Tabla C.2: Esquema de *post-sync* (60 min)

Estos artefactos se emplean en el capítulo de resultados experimentales para comparar perfiles y fases (*sync* vs. *post-sync*).

C.3. Diseño procedimental

Flujo de instalación y preparación

1. **Verificación del entorno**: Se trata de identificar la versión del SO, su kernel, la arquitectura, la CPU, la RAM, el espacio en disco, etc. y algunas advertencias más que mínimas.

2. **Instalación de dependencias**: Para la compilación (`cmake`, `build-essential`), librerías necesarias (`libevent`, `boost`, `sqlite3`, `zmq`) y algunas utilidades (`jq`, `git`, `lsb-release`, `iproute2`, `ca-certificates`).

3. **Usuario y datos**: Creación/verificación de usuario de servicio `Bitcoin` y del `datadir /opt/BLOCKCHAIN` con propiedad `bitcoin:bitcoin`.
4. **Compilación de Bitcoin Core**: Clonado del repositorio oficial, `checkout` v29.0, `cmake` y `install`.
5. **Credenciales**: Generación de `rpcauth` y configuración inicial de `/etc/bitcoin/bitcoin.conf` y `/home/bitcoin/.bitcoin/bitcoin.conf` con las llaves correspondientes generadas.
6. **Servicio**: creación de la unidad `systemd bitcoind.service`, `daemon-reload` y `enable`.

Selección y aplicación de perfiles

El sistema brinda perfiles que configuran `bitcoin.conf` acorde a casos de uso. La lógica de perfil aplica **configuración efectiva**[1] (no narrativa) para asegurar la comparabilidad. La justificación de cada parámetro se encuentra en C.4 (Componentes principales del diseño arquitectónico).

Perfiles (resumen técnico):

- **Estándar**: `txindex=1`[2].
- **Validador**: `prune=0`, `txindex=1`.
- **Analista**: `prune=0`, `txindex=1`, `blockfilterindex=1`.
- **dApps**: `prune=550`, `txindex=0`, `blockfilterindex=0`.
- **Lightning**: `prune=1000`, `txindex=0`, `blockfilterindex=0`, `mempoolexpiry=336`, `maxconnections=40`.
- **Educador**: `prune=1000`, `txindex=0`, `blockfilterindex=0`, `assumevalid=0`[3] , `maxconnections=20`, `mempoolexpiry=72`, `dbcache=100`.

[1] Se entiende por "configuración efectiva" la aplicación real de los parámetros en el archivo bitcoin.conf, de modo que las diferencias entre perfiles se reflejan en la ejecución del nodo y no solo en la descripción documental.

[2] Aunque en *Bitcoin Core* el valor predeterminado de `txindex` es 0, en este proyecto se adopta `txindex=1` como condición base del perfil *Estándar*. Esta decisión, que replica el comportamiento del `script_tfm.sh`, permite habilitar la indexación completa de transacciones y, con ello, la trazabilidad experimental necesaria para comparar métricas entre perfiles con y sin indexado, sin modificar el código fuente.

[3] El parámetro `assumevalid=0` implica verificar todas las firmas históricas y, por tanto, incrementa el tiempo de sincronización en entornos reales; en este caso se mantuvo sin modificación por coherencia con el `script_tfm.sh`, ya que refleja un propósito pedagógico: mostrar el proceso completo de validación, incluso a costa de rendimiento. Este perfil, por tanto, prioriza la transparencia demostrativa sobre la eficiencia temporal.

Captura de métricas

1. **IBD**: Se detecta `initialblockdownload=true` vía `bitcoin-cli`; se muestrea cada 60 s y se registra el CSV de *sync*. Al finalizar la IBD (cambio a `false`), se cierra el fichero.

2. **Post-sync**: Se inicia un segundo registro de 60 minutos, con métricas de sistema (CPU, RAM, swap, disco, red, carga, procesos, *uptime*) al mismo intervalo y en CSV independiente.

C.4. Diseño arquitectónico

Componentes principales

`script_tfm.sh` Es el que gestiona todo el flujo: chequeos, instalación, creación de usuario y del directorio de datos, compilación, `rpcauth`, perfiles, gestión con `systemd` y monitorización (`-log`).

`bitcoind` (servicio) Nodo de *Bitcoin Core* ejecutado como `User=bitcoin`, con `datadir` en `/opt/BLOCKCHAIN` y configuración en `/etc/bitcoin/bitcoin.conf`.

`systemd` Gestión de `bitcoind`, encargado de controlar sus estados: `enable`, `start/stop`, reinicios en fallo y `RuntimeDirectory`[4].

`bitcoin-cli` Interfaz para `getblockchaininfo` y consulta de estado (IBD, progreso, alturas[5]).

`/proc`, `/sys`, utilidades Fuentes de métricas de sistema: CPU, RAM, disco, red, carga, procesos, etc.

`graficos.py` Análisis de CSV y generación de estadísticas y figuras (series y comparativas).

Interacciones

1. **Despliegue**: El script cuenta con la capacidad de instalación de dependencias; la compilación de *Bitcoin Core*; la configuración de `rpcauth` y, finalmente, la creación/activación del servicio de `bitcoind`.

[4]RuntimeDirectory: Directiva de systemd que crea un directorio temporal para el servicio durante su ejecución, destinado a almacenar archivos de estado o PID, y que se elimina automáticamente al detenerse el servicio.

[5]Cuando se habla de "altura" de un bloque, se refiere a la posición dentro de la cadena, contada desde el bloque génesis.

2. **Perfiles**: El script se encarga de la generación del archivo `bitcoin.conf` con los parámetros del perfil elegido sin necesidad de configuración manual por parte del usuario.
3. **Ejecución**: `bitcoind` se ejecuta en background; es el propio `systemd` el que gestiona el ciclo de vida.
4. **Métricas**: El script muestrea el sistema y `bitcoin-cli`, guardando en CSV (*sync* y *post-sync*).
5. **Análisis**: El script `graficos.py` transforma CSV en tablas y gráficos para su incorporación a la memoria.

Trade-offs de diseño

- **Compilación desde fuente**: Maximizar la transparencia y la reproducibilidad (control de versión v29.0; opciones de build).
- **Perfiles parametrizados**: Permiten la comparación de *casos reales de uso* que pueden tener cambios mínimos, sin sesgos manuales.
- **Monitorización ligera**: CSV con separador `";"`, que limita los malentendidos numéricos en `pandas` y permite un análisis sencillo.
- **Sin virtualización**: No altera las métricas (CPU/RAM/IO) y permite reducir la complejidad del entorno.
- **`assumevalid=0` en Educador**: Obliga a la comprobación completa en entornos didácticos, sacrificando un poco de rendimiento para garantizar la consistencia y la claridad experimental.

Riesgos y mitigaciones

- **Variabilidad de hardware**: Mitigado mediante perfiles y muestreo homogéneo de duración 60 segundos.
- **Cambios futuros en *Bitcoin Core***: Fijada para el TFM en `v29.0`; el diseño permite la actualización sin problema con las nuevas versiones de Bitcoin Core.
- **Ruidos de red/sistema**: Pruebas separadas por perfil y por fase (*sync*/*post-sync*), para poder aislar los efectos.

Nota: Los detalles de compilación, instalación, estructura de la entrega y guías de extensión se documentan en la Documentación del Programador. Las instrucciones operativas para usuarios finales se recogen en la Documentación de usuario.

Apéndice D

Documentación del Programador

D.1. Introducción

Este apéndice trata sobre aquellos apartados técnicos para entender, mantener y poder extender la solución software que ha sido conseguida en el TFM. El sistema presente en el TFM consiste en:

- Un **script Bash** (`script_tfm.sh`) que se encarga de la verificación del entorno, la instalación y compilación de *Bitcoin Core* (v29.0), la creación del servicio `systemd`, la aplicación de perfiles de configuración y la monitorización ligera.
- Un **script Python** (`graficos.py`) que lee los CSV generados y produce estadísticas y figuras comparativas para el análisis de resultados.

El enfoque aquí es el **código entero** (estructura, puntos a extender, contrato de datos, convenciones, pruebas técnicas). La operación para los usuarios finales está en la Documentación de usuario.

D.2. Estructura de directorios

La estructura que se muestra a continuación corresponde al **paquete documental del TFM**, estructurado con propósitos de trazabilidad, análisis y reproducibilidad.

En un entorno de ejecución real, un fichero CSV de métricas es producido por el módulo de monitorización del `script_tfm.sh` en el sistema operativo GNU/Linux, dentro del directorio:

```
/var/log/bitcoin-monitor/
```

De este modo, los ficheros `sync_log_{perfil}_{timestamp}.csv` y `post_sync_metrics_{perfil}_{timestamp}.csv` se van creando de forma dinámica, siendo usados posteriormente por `graficos.py` para obtener las figuras y las estadísticas.

Por otro lado, para poder realizar la entrega y la documentación del proyecto, se volvió a replicar esta estructura en un entorno controlado dentro del paquete del TFM, conforme a la siguiente organización:

```
TFM/
  docs/
    csv/
      resumen_estadistico.csv
      resumen_completo.csv
      tabla_comparativa.csv
      estadisticas_globales.csv
      metrics/
        sync/
          sync_log_estandar_*.csv
          ...
        post/
          post_sync_metrics_*.csv
  img/
    metrics/
      sync/
        estandar/cpu_percent.png
        ...
      post/
        validador/mem_used_MB_boxplot.png
        ...
      comparativas/
        comparativa_sync_cpu_percent.png
        comparativa_post_disk_used_MB.png
```

De este modo se garantiza la **trazabilidad entre los datos de ejecución real** y su **reproducción documental**, garantizando que los análisis, gráficos y tablas puedan comprobarse independientemente del entorno de ejecución.

Convenciones de datos. Los CSV de la ejecución **se generan en runtime** en `/var/log/bitcoin-monitor/` con separador `";"` y muestreo cada 60 segundos. En `docs/csv` se agrupan CSV que representan los resultados de campañas de prueba[1] (en la búsqueda de reproducir un análisis sin tener que volver a ejecutar la IBD).

[1]Campaña de prueba: Conjunto planificado de ejecuciones experimentales realizadas bajo condiciones controladas, destinadas a obtener datos comparables entre distintos perfiles o configuraciones.

D.3. Manual del programador

script_tfm.sh: organización y funciones clave

Entrada/salida y requisitos. El script debe ejecutarse con privilegios de `root` (verificación mediante `$EUID`). Instala dependencias (vía `apt-get`) y compila *Bitcoin Core* v29.0 descargado desde la fuente original de GitHub.

Funciones principales (resumen).

- `verificar_root()`,: Validar que se esté ejecutando el script en modo root.
- `verificar_requisitos()`: Chequeo de CPU, RAM, disco, kernel/OS y confirmación interactiva.
- `instalar_dependencias()`: Instalación de herramientas de compilación y utilidades (por ejemplo, `cmake`, `build-essential`, `git`, `lsb-release`, `iproute2`, `jq`, librerías `libevent`, `boost`, `sqlite3`, `zmq`, etc.).
- `crear_usuario_y_directorio()`: Crea el servicio `bitcoin` y el `datadir /opt/BLOCKCHAIN` (propietario bitcoin:bitcoin).
- `compilar_bitcoin()`: Clona el repositorio oficial desde GitHub (`checkout` v29.0), compila con `cmake` (ZMQ opcional) e instala los binarios.
- `generar_rpc_auth()`: Crea la contraseña y la llave con `rpcauth` y vuelca en `/etc/bitcoin/bitcoin.conf` y `/home/bitcoin/.bitcoin/bitcoin.conf` los valores generados de la contraseña y la llave.
- `configurar_servicio()`: Define `/etc/systemd/system/bitcoind.service` con `User=bitcoin`, `Type=forking`, `PIDFile` y `Restart=on-failure`.
- **Perfiles**: `configurar_perfil_estandar()`, `_validador()`, `_analista()`, `_dapps()`, `_lightning()`, `_educador()` - generan `bitcoin.conf` con las configuraciones adecuadas para cada perfil.
- **Monitorización**: `iniciar_monitoreo_log(perfil)` (IBD) y `monitoreo_post_1hora(perfil)` (60 min).

Perfiles (contrato técnico).

- **Estándar**: txindex=1
- **Validador**: prune=0,txindex=1
- **Analista**: prune=0,txindex=1,blockfilterindex=1

- **dApps**: prune=550,txindex=0,blockfilterindex=0
- **Lightning**: prune=1000,txindex=0,blockfilterindex=0,mempoolexpiry=336,maxconnections=40
- **Educador**: prune=1000,txindex=0,blockfilterindex=0,assumevalid=0,maxconnections=20,mempoolexpiry=72,dbcache=100

CSV y muestreo.

- **IBD (sync)**: `sync_log_{perfil}_{timestamp}.csv` con los siguientes campos: `ciclo;timestamp;profile;blocks;headers;verificationprogress;cpu_percent;mem_percent;disk_used_MB;net_rx_kB;net_tx_kB`.
- **Post-sync (60 min)**: `post_sync_metrics_{perfil}_{timestamp}.csv` con CPU, cargas,RAM/swap, disco, red, uptime y procesos (ver esquema en el Apéndice C).

Puntos de extensión

Añadir un nuevo perfil.

1. Crear una función `configurar_perfil_nuevo()` que llame a `preservar_datos_conf()`, que añada los parámetros actuales en un fichero `*.tmp` y que luego haga `mv` sobre `/etc/bitcoin/bitcoin.conf`.
2. Añadir esta opción en `menu_perfiles()` (impresión y `case`).
3. Si cambia el patrón de métrica, ajustar `graficos.py` solo si se añaden columnas *nuevas*.

Cambiar frecuencia de muestreo. Cambiar el `sleep 60` en `iniciar_monitoreo_log` y `monitoreo_post_1hora`, además de actualizar la documentación del esquema si se cambia la ventana efectiva.

Agregar nuevas métricas.

- En **Bash**: Leer nuevas fuentes en cada bucle y añadir las columnas tanto en la *cabecera* como en las *filas*.
- En **Python**: Enseñar a `graficos.py` a cargar y graficar las nuevas columnas (series y comparativas si aplica).

Cambiar `datadir`. Mantener coherencia `datadir=` entre `/etc/bitcoin/bitcoin.conf`, `/home/bitcoin/.bitcoin/bitcoin.conf` y los comandos de medición de disco (`df -m <ruta>`).

Convenciones y estilo

- **Separador CSV**: siempre usar `";"` (pues es robusto frente a `","` y locales distintas).
- **Nombres de archivos**: incluir `{perfil}` y `{timestamp}` para trazabilidad.
- **Permisos**: `/opt/BLOCKCHAIN` propiedad `bitcoin:bitcoin`. Archivos sensibles con `chmod 600` cuando corresponda (por ejemplo, `/home/bitcoin/.bitcoin/bitcoin-.conf`).
- **Usuario de servicio**: `bitcoin` sin privilegios, gestionado por `systemd`.

graficos.py: contrato de entrada y salidas

- **Entrada**: CSV en `docs/csv/metrics/sync/` y `docs/csv/metrics/post/`. Los encabezados deben coincidir con los definidos por `script_tfm.sh`.
- **Salida**: tablas agregadas en `docs/csv/` y figuras en `img/metrics/` (subcarpetas `sync`, `post`, `comparativas`).
- **Robustez**: uso de `pandas` con separador `;` y manejo de conversiones numéricas.

D.4. Compilación, instalación y ejecución del proyecto

Entorno y dependencias

- GNU/Linux x86_64 con `systemd` y privilegios de `root`/`sudo`.
- Acceso a `apt-get` para poder instalar: `build-essential`, `cmake`, `pkgconf`, `python3`, `git`, `lsb-release`, `iproute2`, `ca-certificates`, `jq`, `libevent-dev`, `libboost-dev`, `libsqlite3-dev`, `libzmq3-dev`.

Compilación y despliegue (desarrollador)

```
$ cd TFM/source
$ sudo ./script_tfm.sh
# -> seguir menú: instalación completa (compila v29.0), genera rpcauth,
#    crea servicio systemd y permite aplicar un perfil.
```

Ejecución y métricas

```
# Arranque/Estado del servicio:
$ sudo systemctl start bitcoind
$ systemctl status bitcoind
$ journalctl -u bitcoind -f

# Modo de registro de métricas (IBD + 60 min post-sync):
$ cd TFM/source
$ sudo ./script_tfm.sh -log
```

D.5. Pruebas del sistema

Pruebas de humo (post-instalación)

1. **Binarios**: `bitcoind -version`, `bitcoin-cli -version`.
2. **Servicio**: `systemctl status bitcoind` debe mostrar `active (running)` o progreso de IBD.
3. **RPC**: `bitcoin-cli getblockchaininfo` devuelve JSON válido.

Pruebas de perfiles

Para cada perfil (Estándar, Validador, Analista, dApps, Lightning, Educador):

1. Aplicar el perfil desde el menú (`script_tfm.sh`).
2. Verificar `/etc/bitcoin/bitcoin.conf` con los parámetros esperados.
3. Reiniciar servicio y comprobar `getblockchaininfo`.

Pruebas de monitorización

1. Ejecutar `-log` y confirmar creación de `/var/log/bitcoin-monitor/`.
2. Durante IBD: comprobar crecimiento de `sync_log_*.csv`.
3. Tras IBD: verificar creación de `post_sync_metrics_*.csv` (60 muestras).

Pruebas de análisis (`graficos.py`)

1. Ubicar CSV en `docs/csv/metrics/sync/` y `docs/csv/metrics/post/`.
2. Ejecutar `python3 graficos.py`.
3. Validar aparición de figuras en `img/metrics/` y de tablas en `docs/csv/`.

Pruebas de regresión mínimas

- Cambiar entre perfiles y repetir `-log` para verificar que no se cambian sus cabeceras, su formato, etc.
- Cambiar el `datadir` de forma controlada y comprobar que `disk_used_MB` resulta correcto.
- Realizar prueba sin ZMQ y, opcionalmente, prueba con ZMQ para confirmar que no depende del subsistema de métricas.

Diagnóstico y resolución

- **El Servicio no arranca**: Revisar `journalctl -u bitcoind`, los permisos de la carpeta `/opt/BLOCKCHAIN` (pertenece a `bitcoin:bitcoin`) y la sintaxis de la configuración que se realiza mediante el fichero `/etc/bitcoin/bitcoin.conf`.
- **Los CSV vacíos**: Hay que comprobar que está disponible `bitcoin-cli`, la interfaz de red (`ip route get 8.8.8.8`) y los permisos de la carpeta `/var/log/bitcoin-monitor/`.

Con esta documentación, un desarrollador es capaz de **entender** cómo funciona la solución, pudiendo **modificar** perfiles y métricas sin romper el entorno de compatibilidad y **validar** sus cambios mediante un conjunto mínimo de pruebas reproducibles.

Apéndice E

Documentación de usuario

E.1. Introducción

Este apéndice explica paso a paso las pautas para la **instalación** y **uso** del sistema que se ha desarrollado: un conjunto de scripts que ejecutan el proceso de instalar *Bitcoin Core* (v29.0) a partir del código fuente, aplican **perfiles de uso** determinados de antemano y, además, registran **métricas** de rendimiento durante el *Initial Block Download* (IBD) y 60 minutos posteriores (*post-sync*), entre otros. También se explica la forma en la que se generan las **gráficas** y **tablas** a partir de los CSV.

E.2. Requisitos de usuarios

Conocimientos previos

- Manejo básico de la terminal en GNU/Linux.
- Nociones elementales de `systemd` (iniciar, detener y consultar servicios).

Requisitos de hardware (orientativos)

- CPU con ≥ 2 núcleos (recomendado ≥ 4).
- RAM $\geq 4\,\text{GB}$ (recomendado $\geq 8\,\text{GB}$).
- Almacenamiento:
  - Perfiles **sin poda** (`prune=0`): $\geq 500\,\text{GB}$ libres (recomendado $1\,\text{TB}$).
  - Perfiles **podados** (`prune` $\in$ 550–1000): $\sim$ 30–40 GB.
- Conexión a Internet estable.

Requisitos de software

- GNU/Linux x86_64 con `systemd` y privilegios de `root`/`sudo`.
- Acceso a `apt-get`. El instalador añadirá, si faltan: `build-essential`, `cmake`, `pkg-conf`, `python3`, `git`, `lsb-release`, `iproute2`, `ca-certificates`, `jq`, `libevent-dev`, `libboost-dev`, `libsqlite3-dev`, `libzmq3-dev`.

E.3. Instalación

Repositorio

El código fuente del proyecto se encuentra disponible en el repositorio público:

`https://github.com/joedular/tfm-bitcoin-node`[1]

Contenido del paquete

```
tfm-bitcoin-node/
  docs/
    csv/
      resumen_completo.csv
      resumen_estadistico.csv
      tabla_comparativa.csv
      metrics/
        sync/   # CSV de IBD (por perfil, ejemplo)
        post/   # CSV de post-sync (por perfil, ejemplo)
    logs/       # logs de instalación/ejecución por perfil (entregados)
  img/
    metrics/
      sync/         # figuras por perfil durante IBD (ejemplo)
      post/         # figuras por perfil en post-sync (ejemplo)
      comparativas/ # figuras comparativas entre perfiles (ejemplo)
  source/
    script_tfm.sh   # instalación, perfiles, servicio y monitorización
    graficos.py     # análisis de CSV y generación de figuras/tablas
```

Pasos de instalación (guiada por menú)

1. Clonar el repositorio, acceder a `tfm-bitcoin-node/` y dar permisos de ejecución por medio de la terminal de Linux:

[1] El código y la documentación asociados a este proyecto se publican bajo licencia MIT, permitiendo su libre uso y modificación con atribución al autor.

```
$ git clone https://github.com/joedular/tfm-bitcoin-node.git
$ cd tfm-bitcoin-node/source
```

2. Ejecute el instalador con privilegios de root:

```
$ sudo ./script_tfm.sh
```

3. El instalador realiza los siguientes pasos:
   - Verifica requisitos (CPU, RAM, disco, OS) y luego solicita confirmación para poder continuar.
   - Crea el usuario `bitcoin`, además, el directorio de datos `/opt/BLOCKCHAIN`.
   - Clona y compila *Bitcoin Core* v29.0 (opcionalmente se puede activar `WITH_ZMQ`[2]).
   - Genera credenciales RPC y configura:

     `/etc/bitcoin/bitcoin.conf`
     `/home/bitcoin/.bitcoin/bitcoin.conf`
   - Crea y habilita el servicio `bitcoind` en `systemd`.
   - Permite elegir entre algún **perfil de uso** o la configuración estándar.

Desinstalación (opcional)

Si *Bitcoin Core* ya está instalado, el menú mostrará la opción de desinstalar[3]. Ejecute:

```
$ cd tfm-bitcoin-node/source
$ sudo ./script_tfm.sh
```

y elija *Desinstalar Bitcoin Core* cuando el sistema lo proponga.

E.4. Manual del usuario

Arranque, parada y estado del servicio

```
# Para iniciar el servicio:
$ sudo systemctl start bitcoind
# Para parar el servicio:
$ sudo systemctl stop bitcoind
# Para ver estado del servicio:
$ systemctl status bitcoind
# Para ver logs en tiempo real:
$ journalctl -u bitcoind -f
```

[2]Durante la compilación, el script ofrece opcionalmente activar el flag `-DWITH_ZMQ=ON`, de modo coherente con la configuración de dependencias ZMQ sin intervención manual.

[3]Esta opción borrará todos los ficheros y configuraciones de Bitcoin Core dentro del equipo.

Selección o cambio de perfil

Puede elegir un perfil durante la instalación o cambiarlo posteriormente:

```
$ cd tfm-bitcoin-node/source
$ sudo ./script_tfm.sh
# -> Opción: Cambiar perfil de configuración
```

Perfiles disponibles (resumen funcional):

- **Estándar**: `txindex=1`. Establecido como línea base de estudio con acceso amplio a datos on-chain (en una verdadera configuración estándar, el valor sería 0, pero como se está ocupando para posterior análisis, se define en este caso como 1).
- **Validador**: `prune=0, txindex=1`. Validación completa (alto uso de disco).
- **Analista**: `prune=0, txindex=1, blockfilterindex=1`.
- **dApps**: `prune=550, txindex=0, blockfilterindex=0`. Ligero para desarrollo.
- **Lightning**: `prune=1000, txindex=0, blockfilterindex=0, mempoolexpiry=336, maxconnections=40`. Backend para LN.
- **Educador**: `prune=1000, txindex=0, blockfilterindex=0, assumevalid=0, maxconnections=20, mempoolexpiry=72, dbcache=100`.

Comprobar progreso de sincronización

```
$ bitcoin-cli getblockchaininfo
```

Revise `blocks`, `headers`, `verificationprogress` e `initialblockdownload`.
Cuando `initialblockdownload=false`, la IBD ha finalizado.

Registro de métricas (modo `-log`)

Para registrar métricas durante la IBD y, al finalizar, 60 minutos adicionales:

```
$ cd tfm-bitcoin-node/source
$ sudo ./script_tfm.sh -log
```

Se crearán CSV en `/var/log/bitcoin-monitor/`:

- `sync_log_{perfil}_{timestamp}.csv` (IBD cada 60 segundos hasta que logre sincronizar completamente)
- `post_sync_metrics_{perfil}_{timestamp}.csv` (60 muestras, 60 min)

Generar figuras y tablas desde CSV

Puede usar los CSV propios o los de ejemplo incluidos:

1. Ubique los CSV en:

   ```
   tfm-bitcoin-node/docs/csv/metrics/sync/
   tfm-bitcoin-node/docs/csv/metrics/post/
   ```

   Y luego copie todos los CSV en una carpeta `TFM/source/bitcoin-monitor/`.

2. Ejecute:

   ```
   $ cd tfm-bitcoin-node/source
   $ python3 graficos.py
   ```

   Nota: Se debe modificar el directorio dentro de graficos.py agregando la carpeta `TFM/source/bitcoin-monitor/`.

3. Resultados esperados:

   - Tablas: `TFM/source/bitcoin-monitor/graficos/resumen_estadistico.csv`, `TFM/source/bitcoin-monitor/graficos/comparativas/resumen-_estadistico.csv`, `TFM/source/bitcoin-monitor/graficos/comparativas/tabla_comparativa.csv`
   - Figuras: `TFM/source/bitcoin-monitor/graficos/sync/`, `TFM/source/bitcoin-monitor/graficos/post/`, `TFM/source/bitcoin-monitor/graficos/comparativas/`

Ubicación de datos y configuraciones

- **Datos de la cadena**: `/opt/BLOCKCHAIN` (propietario: `bitcoin:bitcoin`).
- **Config. global**: `/etc/bitcoin/bitcoin.conf`.
- **Config. usuario servicio**: `/home/bitcoin/.bitcoin/bitcoin.conf`.
- **CSV generados en ejecución**: `/var/log/bitcoin-monitor/`.
- **Logs entregados para TFM**: `TFM/docs/logs/`.

Resolución de problemas (FAQ breve)

- **El servicio no arranca.** Revise:

  ```
  $ systemctl status bitcoind
  $ journalctl -u bitcoind -xe
  ```

  Verifique permisos de `/opt/BLOCKCHAIN` (`chown bitcoin:bitcoin /opt/BLOCKCHAIN`) y la sintaxis de `/etc/bitcoin/bitcoin.conf`.

- **Falta de espacio.** Use un perfil podado (`dApps`, `Lightning`, `Educador`) o amplíe almacenamiento.

- **No se generan CSV con `-log`.** Asegure que `bitcoin-cli` responde, existe `/var/log/bitcoin-monitor/` y hay permisos de escritura.

- **Quiero cambiar `datadir`.** Mantenga coherencia en ambos ficheros: `/etc/bitcoin/bitcoin.conf` y `/home/bitcoin/.bitcoin/bitcoin.conf`.

Buenas prácticas

- Mantenga el sistema actualizado (`apt update && apt upgrade`).
- Evite apagar el equipo durante la IBD.
- Elija el perfil según el caso de uso: **Analista/Validador** para análisis completo, **Educador/dApps** para docencia y prototipos, **Lightning** como backend de LN.

Bibliografía

[1] Acuña, H. Estudio sobre bitcoin y tecnología blockchain. *Cuadernos CEF*, 1 (2017).

[2] Allende, S. L., Gibellini, F. A., Sánchez, C. B., and Serna, M. M. *Sistemas operativos: Linux teoría y práctica*, 2 ed. edUTecNe, Ciudad Autónoma de Buenos Aires, 2019. Libro digital, PDF.

[3] Alsahan, L., Lasla, N., and Abdallah, M. Local bitcoin network simulator for performance evaluation using lightweight virtualization. *arXiv preprint arXiv:2002.01243* (2020).

[4] Antonopoulos, A. M. *Mastering Bitcoin: Programación de la cadena de bloques abierta*, 1 ed. Independently published, Estados Unidos, 2019. Traducción al español de *Mastering Bitcoin: Programming the Open Blockchain*.

[5] Antonopoulos, A. M., and Harding, D. A. *Mastering Bitcoin: Programming the Open Blockchain*, 3 ed. O'Reilly Media, Sebastopol, CA, 2023.

[6] Bartolomé, A., and Moral Ferrer, J. M., Eds. *Blockchain en Educación: Cadenas rompiendo moldes*. Learning, Media & Social Interactions / Universitat de Barcelona, Barcelona, 2018.

[7] Campbell-Verduyn, M., Ed. *Bitcoin and Beyond: Cryptocurrencies, Blockchains, and Global Governance*. Routledge, Abingdon, UK and New York, USA, 2018. Open Access version available.

[8] Champagne, P. *El libro de Satoshi*. e53 Publishing LLC / Blockchain España, Madrid, 2014.

[9] Developers, B. C. Bitcoin core 29.0 release notes, 2024. [Internet; accedido 14-julio-2025].

[10] Drescher, D. *Blockchain Basics: A Non-Technical Introduction in 25 Steps*. Springer, Cham, Switzerland, 2017.

[11] Fan, C., Ghaemi, S., Khazaei, H., and Musilek, P. Performance evaluation of blockchain systems: A systematic survey. *IEEE Access 8* (2020), 126927–126950.

[12] Hearn, M. Bip 37: Connection bloom filtering, 2012. [Bitcoin Improvement Proposal].

[13] Learn, G. Nodos de bitcoin vs. mineros: Diferencias clave explicadas, 2024. [Internet; accedido 8-septiembre-2025].

[14] Lim, C. L., and Janse, A. *Blockchain Handbook*. De Boekdrukker, Amsterdam, 2020.

[15] Medina, M. G. Un estudio del rendimiento del minado bitcoin en escenarios de merged mining. Tesis de maestría, Universidad de Buenos Aires, Buenos Aires, Argentina, 2021.

[16] Osuntokun, O., Akselrod, A., and Posen, J. Bip 157: Client side block filtering, 2017. [Bitcoin Improvement Proposal].

[17] Prypto. *Bitcoin For Dummies*. John Wiley & Sons, Hoboken, NJ, USA, 2016.

[18] Pérez Conte, R. Bitcoin: Identificar nodos mineros. Master's thesis, Universitat Oberta de Catalunya (UOC), Máster Interuniversitario de Seguridad de las Tecnologías de la Información y de las Comunicaciones (MISTIC), Sabadell, España, 2015.

[19] Riquet, A., Ngo, V., and Doyen, L. Impact of network topologies on blockchain performance: The lilith framework. In *Middleware '21: Proceedings of the 22nd International Middleware Conference: Demos and Posters* (2021), ACM, pp. 5–6.

[20] SUSE LLC. *Introducción a los conceptos básicos de systemd*, Sept. 2025. [Internet; accedido 3-agosto-2025].

[21] Tschorsch, F., and Scheuermann, B. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys & Tutorials 18*, 3 (2016), 2084–2123.

[22] Wang, C., Chu, X., and Yang, Q. Measurement and analysis of the bitcoin networks: A view from mining pools. *arXiv preprint arXiv:1902.07549* (2019).

[23] Wright, C. S. The redundancy of full nodes in bitcoin: A network-theoretic demonstration of miner-centric propagation topologies. *arXiv preprint arXiv:2506.14197* (2025).

[24] Wuille, P., Osuntokun, O., and Akselrod, A. Bip 158: Compact block filters for light clients, 2017. [Bitcoin Improvement Proposal].

[25] Zabka, P., Foerster, K.-T., Schmid, S., and Decker, C. Empirical evaluation of nodes and channels of the lightning network. *Pervasive and Mobile Computing 83* (2022), 101584.

[26] Álvarez Pizarro, Y. A., Sánchez Galvis, I. J., and Zabala Vargas, S. A. Estudio comparativo para la selección de una red blockchain pública para notificaciones certificadas en organizaciones gubernamentales. In *XX Congreso Latino-Iberoamericano de Gestión Tecnológica y de la Innovación (ALTEC)* (Paraná, Argentina, 2023), Asociación Latino-Iberoamericana de Gestión Tecnológica.