



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN.

MÁSTER EN INGENIERÍA DE TELECOMUNICACIONES

ÁREA DE INGENIERÍA TELEMÁTICA

TRABAJO FIN DE MÁSTER

**INTEROPERABILIDAD ENTRE NODOS GETH Y BESU EN REDES
ETHEREUM PERMISIONADAS. APLICACIÓN EN LA RED T DE
ALASTRIA**

**D. Rodríguez Fernández, Javier
TUTOR: D. Nuño Huergo, Pelayo
COTUTOR: D. De la Vega Sánchez, Ignacio**

FECHA: Julio 2020

Índice general

1. Introducción	1
1.1. Descripción del proyecto	1
1.2. Objetivo	2
1.3. Motivación	3
2. Background Tecnológico	4
2.1. Qué es blockchain	4
2.1.1. Origen	5
2.1.2. Principios básicos del blockchain	6
2.2. Funcionamiento	7
2.2.1. Nodos de la red	7
2.2.2. Formación de la cadena de bloques	9
2.2.3. Generación de nuevos bloques	12
2.2.3.1. PoW - Proof of Work	13
2.2.3.2. PoA - Proof of Authority	13
2.2.3.3. PoS - Proof of Stake	14
2.2.3.4. Istanbul Byzantine Fault Tolerant (IBFT)	15
2.3. Conceptos relacionados con la tecnología blockchain	17

2.3.1. Dificultad	18
2.3.2. Recompensas	18
2.3.3. Smart Contracts	19
2.3.4. Dapp - Aplicaciones distribuidas	20
2.4. Tipos de redes de blockchain	20
2.4.1. Redes blockchain pública	21
2.4.2. Redes blockchain privadas	21
2.4.3. Redes de blockchain permisionadas	21
2.5. Principales sectores de aplicación de blockchain	22
2.6. Inconvenientes de la tecnología de blockchain	23
3. Hipótesis de partida y Alcance del proyecto	24
3.1. Ethereum	24
3.1.1. Criptomoneda de Ethereum	25
3.1.2. Gas en la red Ethereum	25
3.1.2.1. Qué es el Gas	26
3.1.2.2. Utilidades principales del Gas	26
3.1.2.3. Uso del Gas como recompensa	27
3.1.3. Inconvenientes de Ethereum	27
3.2. Quorum	28
3.2.1. Características de Quorum	28

3.2.1.1.	Tipos de permisionado en Quorum	28
3.2.1.2.	Algoritmos de consenso en Quorum	29
3.2.2.	Aplicación de Quorum	30
3.3.	Alastria	30
3.3.1.	Características generales	30
3.3.1.1.	Origen de Alastria	31
3.3.1.2.	Alastria en la actualidad	31
3.3.1.3.	Objetivo de Alastria	31
3.3.1.4.	Ejemplos de casos de uso	32
3.3.2.	Características técnicas	33
3.3.2.1.	Tipo de red utilizada	33
3.3.2.2.	Red T	34
3.3.2.3.	Red B	35
3.3.3.	Objetivos futuros de Alastria	36
4.	Trabajo realizado	38
4.1.	Entorno de desarrollo	38
4.2.	Instalación de la Red T	39
4.2.1.	Nodos de la red T	42
4.2.1.1.	Enode	43
4.2.1.2.	Cliente Geth	44

4.2.2. Configuración de la red	46
4.2.2.1. Script: start_network	46
4.2.2.2. Script: start_node	49
4.2.3. Resumen funcionamiento Red T	51
4.3. Instalación de la Red B	55
4.3.1. Procedimiento de instalación Red B	56
4.3.2. Cliente Hyperledger Besu	57
4.3.3. Configuración de la Red B	59
4.3.4. Resumen de funcionamiento de la Red B	64
4.4. Interoperabilidad entre cliente Geth y Besu	70
4.4.1. Parámetros fundamentales a considerar	71
4.4.1.1. Análisis del origen de la red	71
4.4.1.2. Identificador de la red	72
4.4.1.3. Permisionado de la red	73
4.4.1.4. Conexión entre máquinas virtuales	73
4.4.2. Procedimiento seguido	73
4.4.2.1. Conexión de nodos sin permisionado	74
4.4.2.2. Conexión de nodos con permisionado	82
4.4.2.3. Integración de nodo Besu en la Red T de Alastria	84
5. Planificación	91

6. Conclusiones	94
6.1. Conclusiones técnicas	94
6.2. Conclusiones personales	95
7. Líneas futuras	96
Bibliografía	99

Índice de figuras

2.1. Esquema de tipos de nodos en una red	9
2.2. Representación de la relación entre bloques	10
2.3. Representación de la idea del Árbol de Merkle	11
2.4. Esquema de intento de modificación de un bloque	12
2.5. Diagrama de estados del algoritmo de consenso IBFT	17
3.1. Esquema con la misión y visión de Alastria	32
3.2. Esquema de tipo de red	34
3.3. Esquema de tipo de red	37
4.1. Árbol de directorios utilizado para ejecutar la Red T	42
4.2. Ejecución de Geth mediante la línea de comandos	44
4.3. Inicio de la consola interactiva de Geth	45
4.4. Consulta en la consola interactiva de Geth	46
4.5. Fragmento del script start_network	47
4.6. Diagrama de funcionamiento de IBFT	48
4.7. Script start_node: Declaración de variables	49
4.8. Script start_node: Invocación nodo validador	51
4.9. Script start_node: Invocación nodo regular	51

4.10. Ejemplo de una red con tres nodos	52
4.11. Información de los nodos de la red (1)	53
4.12. Información de los nodos de la red (2)	53
4.13. Información sobre el bloque 1743	54
4.14. Información sobre el bloque 1744	55
4.15. Información sobre el bloque 1745	55
4.16. Línea de comandos del cliente Besu	58
4.17. Línea de comandos del cliente Besu	58
4.18. Ejemplo de un archivo <i>genesis.json</i>	59
4.19. Script para ejecutar el Nodo 1 de la Red B	61
4.20. Script para ejecutar los nodos 2, 3 y 4 de la Red B	64
4.21. Red B- Número de nodos conectados	65
4.22. Red B- Información sobre los nodos	66
4.23. Red B- Información sobre Id de la Red	66
4.24. Red B- Nuevos bloques y transacciones	67
4.25. Creación de nueva red en Metamask	68
4.26. Historial de transacciones desde Metamask	69
4.27. Información del bloque que contiene una transacción	70
4.28. Información de un bloque que no contiene transacciones	70
4.29. Arquitectura de la red utilizada	74
4.30. Puesta en funcionamiento del bootnode	75

4.31. Script para automatización de los nodos del cliente Geth	75
4.32. Genesis.json	77
4.33. Información del nodo 1 implementado con Geth	78
4.34. Información del nodo 2 implementado con Geth	78
4.35. Información sobre los nodos conectados	78
4.36. Script para automatización del nodo del cliente Besu	79
4.37. Información del nodo implementando con el cliente Besu	80
4.38. Nodos vecinos del nodo Besu	80
4.39. Nodos vecinos de uno de los nodos Geth	81
4.40. Nodos vecinos del otro de los nodos Geth	81
4.41. Nodos Geth sin encontrar nodos vecinos	82
4.42. Archivo <i>permissioned-nodes.json</i> con un enode añadido	83
4.43. Nodo conectado incluido en el permisionado	83
4.44. Archivo <i>permissioned-nodes.json</i> con dos enodes añadidos	84
4.45. Nodos conectados al nodo que tiene el permisionado activo	84
4.46. Inicio de la cadena de bloques	85
4.47. Script start_node.sh modificado	86
4.48. Error con el inicio de la cadena	86
4.49. Inicio del nodo de Besu	87
4.50. Gráfico de los nodos conectados	88
4.51. Información del nodo Main	89

4.52. Información del nodo valitador1	89
4.53. Información del nodo implementado con Besu	90
5.1. Diagrama de Gantt	93

Índice de tablas

5.1. Tareas del proyecto	92
------------------------------------	----



1. Introducción

En este primer capítulo se describen las características más básicas del proyecto, así como el objetivo principal que se pretende cumplir y la motivación por la que surge.

1.1.- Descripción del proyecto

Este proyecto se centra en el estudio y análisis de las principales redes de blockchain y las diferentes tecnologías que permiten implementar dichas redes. Se pretende estudiar las características técnicas, de desarrollo e implementación de estas tecnologías con el fin de poder comprender su funcionamiento para, posteriormente, estudiar la interoperabilidad de estas tecnologías desde el punto de vista de su aplicación en un entorno empresarial.

La primera parte de la investigación realizada en este proyecto aborda los principales tipos redes de blockchain que existen en la actualidad y las diferentes tecnologías que permiten implementar redes de blockchain. En la actualidad, las redes de blockchain se pueden categorizar desde múltiples puntos de vista. No obstante, para el análisis realizado en este proyecto, se analiza el entorno para el que está pensada cada una de estas redes de blockchain. Por una parte, se encuentran las redes pensadas para un entorno público (por ejemplo red Ethereum) y por otra las que están diseñadas para un entorno Empresarial (Quorum).

En segundo lugar, la investigación realizada en este proyecto analiza dos de las tecnologías que permiten implementar las redes de blockchain mencionadas anteriormente. Estas tecnologías son los clientes Geth (Go-Ethereum) y Besu. Estos dos clientes presentan características diferentes pero ambos tienen un mismo propósito: permitir la implementación de redes de blockchain. Durante el desarrollo de este proyecto se estudia cómo funciona cada uno de estos clientes de manera independiente y, posteriormente, se valora la posibilidad de integrar ambos clientes en una única red de blockchain. Unificar el funcionamiento de las dos tecnologías en una misma red



homogénea ofrece beneficios desde el punto de vista de poder obtener ventajas de las mejores características de cada una de ellas, sin necesidad de tener que trabajar con redes diferentes. Además, esto también posibilita evitar el hecho de estar ligado a una única tecnología en concreto que, generalmente, es un problema desde el punto de vista del desarrollo o implementación de cualquier tipo de aplicación.

1.2.- Objetivo

Tras haber descrito las principales ideas del proyecto , se definen los objetivos que se pretenden conseguir con el desarrollo del mismo.

El primer objetivo es conseguir un alto nivel de familiarización y conocimiento de todo lo que rodea a la tecnología blockchain. La idea es analizar los principios y conceptos más importantes que sirven para entender cuáles son las principales ventajas que tiene el blockchain frente a otras tecnologías. Algunos de estos conceptos destacables son la descentralización de la información, la inmutabilidad del registro de operaciones realizadas o la seguridad en general que ofrece a todas las aplicaciones que se ejecutan sobre él.

Como segundo punto de investigación en este proyecto, se plantea el estudio de las diferentes redes de blockchain en la actualidad y que son las más utilizadas. De estas redes, se pretende estudiar cómo funciona cada una y cuáles son sus características propias y que las diferencian de las demás. Una vez estudiado esto, se pretende profundizar en aquellas redes que estén más ligadas con el proyecto Alastria, el cuál es uno de los elementos clave de este proyecto.

Finalmente, la línea de investigación más importante que incluye este proyecto es la comparación de dos clientes, como son Geth y Besu, los cuales permiten implementar las redes de blockchain mencionadas anteriormente. Estos dos clientes son usados en la actualidad en el consorcio de blockchain Alastria, y por dicho motivo, son la parte fundamental de este proyecto. El objetivo principal respecto a estos dos clientes es estudiar su interoperabilidad para poder integrarlos en una única red. Esta última fase de la investigación es la que presenta un carácter más práctico, ya que para poder

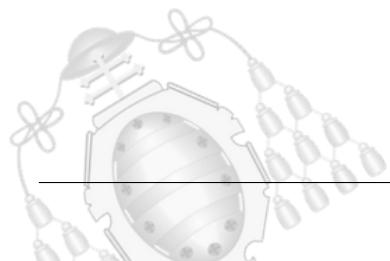


conocer correctamente cómo funciona cada uno de estos clientes, es necesario probar su funcionamiento y todas las opciones de configuración que presentan.

1.3.- Motivación

En los últimos años, la tecnología de blockchain ha ido ganando más peso en todas las noticias de actualidad tecnológica, ya que cada vez surgen más ideas o más alternativas de negocio donde resulta interesante aplicar los beneficios del blockchain. Pero a pesar de este reciente auge, es una tecnología que aún no ha alcanzado el máximo punto de relevancia dentro de la sociedad actual. Sin embargo, se piensa que en los próximos años puede llegar ese momento y convertirse entonces en una de las principales revoluciones tecnológicas de las últimas décadas, además, de convertirse en una de las bases de la industria 4.0. Por dicha razón, se considera interesante profundizar en esta tecnología y en todas las posibles variantes que existen. Además, a pesar de no ser aún una tecnología demasiado extendida en la sociedad, muchas empresas han empezado a apostar por ella para mejorar parte de los servicios que ofrecen. Un ejemplo de esto, es el consorcio Alastria, que se explica en capítulos posteriores.

Alastria tiene como uno de sus principales objetivos de futuro ofertar una red de blockchain homogénea, en la que se integren diferentes clientes en una única red. Por dicha razón, este proyecto se considera interesante para poder analizar y valorar la viabilidad de llevar a cabo la misión de futuro que tiene el consorcio de blockchain Alastria.





2. Background Tecnológico

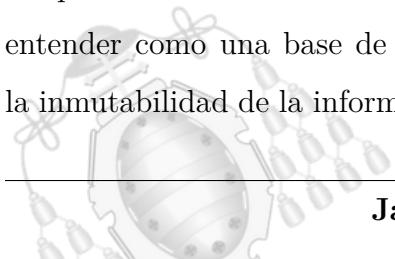
Este capítulo está destinado a comentar todos los conceptos principales sobre la tecnología de blockchain y, especialmente, sobre todo aquellos que son esenciales para poder entender cómo funciona esta tecnología.

2.1.- Qué es blockchain

El término blockchain (cadena de bloques en castellano) no representa únicamente a una tecnología en concreto, sino que este concepto se interpreta como un paradigma que engloba nuevas ideas sobre cómo afrontar el funcionamiento de muchos servicios de la vida cotidiana. Por dicho motivo, es muy difícil poder encontrar una definición concreta del término blockchain. De todas formas, durante este apartado se aportan una serie de definiciones que permiten entender, de una forma sencilla, las ideas principales del funcionamiento de blockchain.

Inicialmente, para entender el concepto de blockchain desde un punto de vista simple, se realiza un símil que deja ver lo que representa la idea de blockchain a rasgos generales. Una cadena de bloques se puede entender como un gran libro de contabilidad en el que se anotan todas los movimientos que se realizan. Cada capítulo de este libro de contabilidad se puede asemejar a cada uno de los bloques de la cadena de bloques. Dentro de cada uno de estos capítulos del libro de contabilidad, se encuentran todos los movimientos o transacciones realizados durante el período de tiempo en el que el capítulo del libro de contabilidad ha estado abierto. Entonces, siguiendo la analogía con la cadena de bloques, dentro de cada uno de los bloques están todas las transacciones realizadas en el tiempo que tarda en generarse y validarse cada uno de los bloques.

La segunda definición que se puede aportar sobre el blockchain, y esta vez desde un punto de vista más técnico, es la siguiente: una cadena de bloques se puede entender como una base de datos descentralizada que presenta una gran ventaja en la inmutabilidad de la información almacenada, es decir, una vez que las transacciones





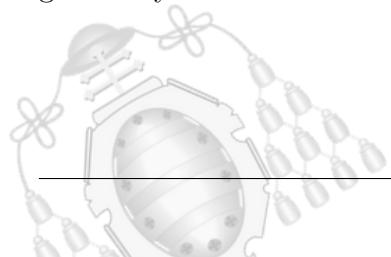
son registradas dentro de la cadena de bloques, esas transacciones ya no se pueden modificar. Esto permite que toda la información almacenada en la cadena de bloques sea totalmente confiable, convirtiéndose esto en una de las grandes fortalezas de esta tecnología.

De la segunda definición aportada, es importante resaltar dos conceptos claves, descentralizado e inmutable. Estos son dos de los pilares básicos de la tecnología de blockchain y que en apartados posteriores son comentados de manera más extensa, pero es esencial tener en cuenta, desde este punto del documento, que estos dos conceptos son la base sobre la que se cimienta la tecnología de blockchain.

2.1.1.- Origen

El origen de la tecnología blockchain se remonta a la década de los años 90. Se considera a Stuart Haber y W. Scott Stornetta [1] como los creadores de la primera idea de la tecnología blockchain. Su idea era desarrollar un sistema que permitiese gestionar y almacenar los documentos digitales en el que nadie tuviese la posibilidad de alternar la marca temporal de la información, de forma que siempre se pudiese tener controlado el orden cronológico de cada uno de los documentos almacenados. Estos mecanismos de seguridad estaban basados en técnicas criptográficas. Todos estos documentos se almacenaban en bloques, los cuáles formaban la cadena de bloque. Posteriormente, para mejorar la eficiencia en el almacenamiento de esos documentos se incorpora un sistema de árbol de Merkle [2] que permite aumentar el número de transacciones incluidas dentro de cada bloque, así como mejorar la seguridad en la información almacenada.

A pesar de que el origen del blockchain se haya marcado en la década de los 90, no fue hasta 2008, con la aparición de la criptomonedra Bitcoin, cuando se empieza a hablar realmente de la tecnología de blockchain. Posteriormente, en 2013, con la aparición de la red Ethereum (uno de los temas principales de este proyecto) el concepto de blockchain gana mayor relevancia.





2.1.2.- Principios básicos del blockchain

En esta sección se describen los principios básicos en los que se fundamenta la tecnología de blockchain. Estos principios son los siguientes [3]:

- Sistema distribuido
- Integridad e inmutabilidad de la información
- Mecanismos de consenso para tomar decisiones en la red

La tecnología de blockchain está basada en la manera de funcionar de los sistemas distribuidos. Esto quiere decir que no existe un elemento central encargado de gestionar toda la red, sino que son los propios nodos, compartiendo información entre ellos, los que son los encargados de tomar decisiones, es decir, permite gestionar toda la información almacenada en una cadena de bloques sin necesidad de una entidad central encargada de supervisar todos los movimientos que se puedan realizar. En este caso son los propios nodos de la red de blockchain los que se encargan de controlar que todo está funcionando correctamente. Además, indirectamente, esta característica mejora la seguridad de las redes de blockchain, ya que al no tener ninguna entidad central que controle todo el flujo de la información, se eliminan los puntos críticos de la red, los cuales suelen ser objetivos de ataques ciberneticos.

Como segunda gran característica de las redes de blockchain destaca la integridad de la información. Esto significa que una vez que la información es registrada y almacenada en la cadena de bloques, dicha información no va a poder ser modificada por ningún usuario. Esto hace que la información almacenada sea totalmente legítima y evita posibles intentos de fraude o de falsificación de la información. Un claro ejemplo donde esto tiene gran valor es el mundo de las criptomonedas, ya que evita posibles fraudes a la hora de realizar las transacciones entre diferentes usuarios. Esta integridad de la información es conseguida gracias a la aplicación de árbol de Merkle al contenido de cada uno de los bloques. Esta técnica se explica más en detalle en apartados posteriores donde se comenta con exactitud cómo funciona realmente una cadena de bloques. Pero lo que permite es que una vez que se ha formado el bloque y se ha añadido a la cadena, si se intenta modificar la información contenida en dicho bloque, el bloque se vería



modificado y ya no encajaría en su posición dentro de la cadena, por lo cuál se podría detectar que ese bloque ha sido manipulado.

Como tercer principio importante de la tecnología de blockchain, se destacan los algoritmos de consenso. Esta es la forma que tienen las redes de blockchain de tomar decisiones para añadir nuevos bloques a la cadena. Como se ha mencionado anteriormente, en este tipo de tecnología los nodos que forman la red son los encargados de tomar las decisiones, sin necesidad de utilizar una entidad central encargada de gestionar todo. Para tomar estas decisiones se utilizan los mecanismos de consenso. Según el tipo de red de blockchain, se utilizan diferentes algoritmos para tomar esas decisiones, algunos de los más habituales son [4]:

- Proof of Work (PoW)
- Proof of Stack (PoS)
- Proof of Authority (PoA)
- Istanbul Byzantine Fault Tolerant (IBFT)

Cada una de estas técnicas utiliza un procedimiento diferente con el que permite validar, entre todos los nodos, los bloques que se van a añadir a la cadena de bloques. Según el tipo de red de blockchain, es más conveniente que se utilice un tipo de algoritmo u otro, pero siempre, toda red de blockchain debe tener configurado algún mecanismo de consenso entre sus nodos para poder validar los bloques que se añaden a la red.

2.2.- Funcionamiento

En esta sección, se describe, desde un punto de vista genérico, el funcionamiento de las redes de blockchain y las técnicas que se utilizan para cumplir los principios anteriormente descritos.

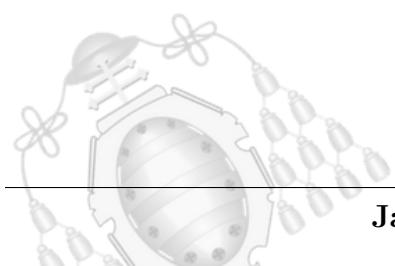
2.2.1.- Nodos de la red

Las redes de blockchain siguen la idea de los sistemas distribuidos. En estos sistemas no existe una entidad central encargada de gestionar todo el flujo de información, sino



que son los propios nodos de la red los encargados de gestionar todos los movimientos. Por dicha razón, en las redes de blockchain se plantea la comunicación P2P (Peer-to-Peer) entre los nodos de la misma. Esto implica la necesidad de que los nodos se puedan conectar unos con otros ya que deben estar sincronizados para tomar las decisiones de la red. Además, para poder reconocer a los nodos en la red y realizar la posterior sincronización, cada nodo debe usar algún protocolo que le permita detectar a los nodos vecinos. Según el tipo de red de blockchain, el tipo de protocolo usado para la detección de otros nodos y para la sincronización puede variar.

También es importante mencionar que dentro de una red de blockchain, los nodos pueden ejecutar diferentes funciones y por dicha razón reciben diferentes nombres. Generalmente, en las redes se pueden distinguir tres tipos de nodos: Validadores, regulares y bootnodes. En la figura 2.1 se muestra un esquema de los tipos de nodos que forman las redes. El primer tipo, son aquellos nodos que tienen mayor responsabilidad dentro de la red, ya que son los encargados de tomar las decisiones de la red. El segundo tipo se refiere a los nodos que se conectan a la red y que tienen una copia de la cadena de bloques pero no se encargan de tomar decisiones. Y el tercer tipo son aquellos nodos que son utilizados por el resto de nodos para detectar a los nodos vecinos y que ayudan en la sincronización. En otros tipos de redes, los nodos validadores también pueden recibir el nombre de mineros.



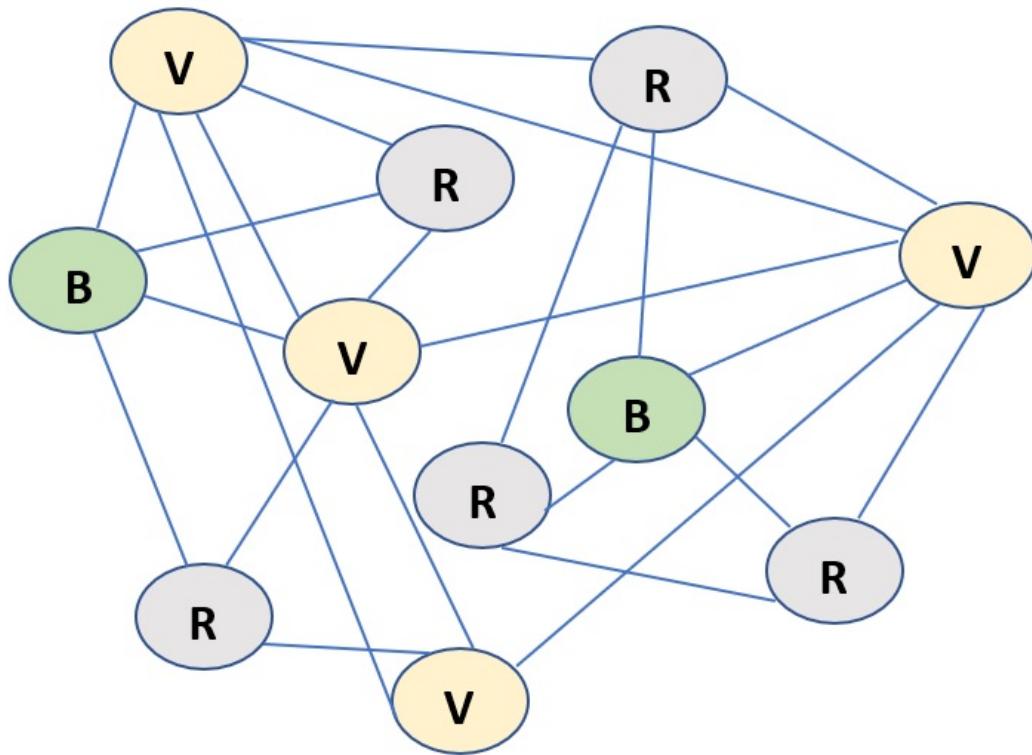


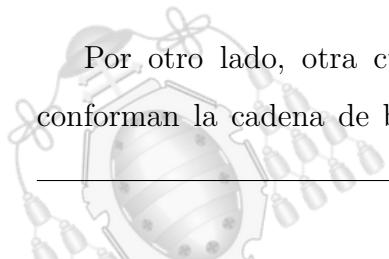
Figura 2.1.- Esquema de tipos de nodos en una red

2.2.2.- Formación de la cadena de bloques

En esta sección se comenta cómo se forma la cadena de bloques y qué mecanismos y técnicas se utilizan para que los bloques vayan ordenados dentro de la cadena. Estos mecanismos son los que también permiten que se cumpla la integridad e inmutabilidad de la información registrada.

Primero, es interesante recordar que el tamaño de cada uno de los bloques es variable en función del tipo de cadena con la que se esté trabajando. En cada uno de estos bloques se registra toda la información de movimientos que se realiza en un período de tiempo determinado. Por ejemplo, el caso de las bitcoins, se genera un nuevo bloque aproximadamente cada 10 minutos, ese bloque tiene una tamaño de unos 2 MB. Dentro de cada bloque se incluyen las transacciones que han sido validadas en el período de tiempo en el que se ha formado dicho bloque.

Por otro lado, otra cuestión que se debe comentar respecto a los bloques que conforman la cadena de bloques, es la forma en la que están relacionados unos con



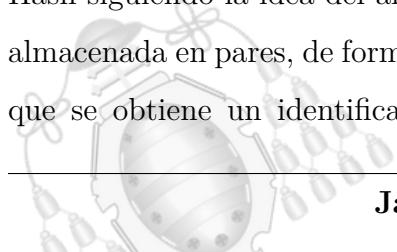


otros. Esto se consigue gracias al uso de la operación criptográfica Hash. En particular, con la tecnología de blockchain se utiliza la función SHA-256. Esta operación permite obtener una cadena única a partir de una entrada de datos [5]. En el caso de los bloques de la cadena, la operación de Hash se aplica a toda la información contenida dentro de ese bloque, de forma que se obtenga un identificador único para cada bloque. Ese identificador único de cada bloque es lo que permite relacionar un bloque con el anterior y el siguiente. Este identificador está compuesto por una cadena de 64 caracteres expresados en lenguaje hexadecimal. Una representación esquemática de este procedimiento se muestra en la figura 2.2 . En esta figura se aprecia como dentro de cada bloque se encuentra el Hash del bloque anterior y el Hash del bloque actual. Esto permite que dentro de cada uno de los bloques haya una referencia del bloque anterior, de forma que si alguno de los bloques pierde esa referencia, la cadena es errónea ya que ese bloque no está enlazado con el bloque anterior y siguiente.



Figura 2.2.- Representación de la relación entre bloques

Hasta ahora se ha comentado la idea más simple de como se enlazan los bloques de la cadena usando el Hash generado a partir del contenido registrado en cada uno de los bloques. No obstante, es necesario profundizar en cómo se usa exactamente la función Hash dentro de la tecnología de blockchain, ya que es otro de los elementos clave que permiten cumplir uno de los principios básicos de esta tecnología, la inmutabilidad y la integridad de la información. Para conseguir esto, se aplica la función Hash, pero no se aplica un único Hash a todo el contenido del bloque, sino que se realizan operaciones Hash siguiendo la idea del árbol de Merkle [2]. Esto consiste en dividir la información almacenada en pares, de forma que se aplican funciones Hash a cada par. De tal manera que se obtiene un identificador para cada par de Hash y esto se realiza de forma





sucesiva hasta tener un único Hash identificativo para todo el bloque. Se muestra una representación esquemática de este proceso en la figura 2.3 .

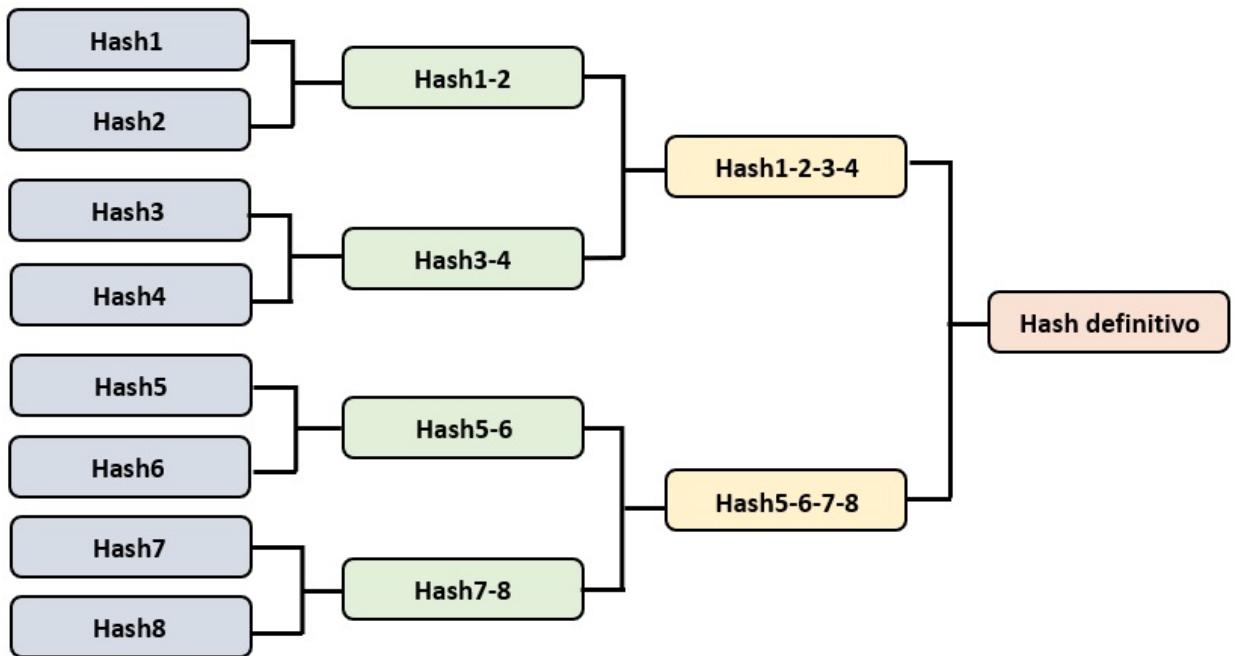
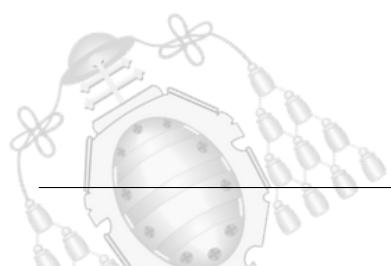


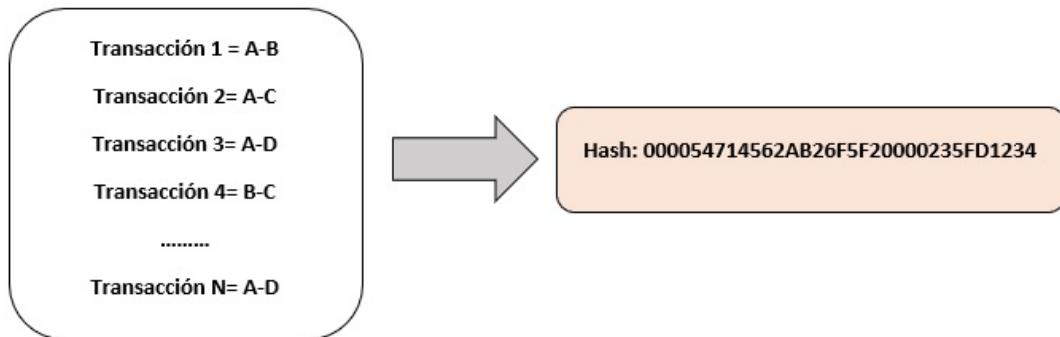
Figura 2.3.- Representación de la idea del Árbol de Merkle

Con este procedimiento se consigue la integridad y la inmutabilidad de la información, ya que como se ha dicho, la operación Hash devuelve un identificador único para una entrada de datos concreta. Si alguno de los datos se modifica, el Hash correspondiente a esos datos también se modifica y por tanto, el Hash final, que identifica a cada bloque, también se modifica. Esto se representa de manera esquemática en la figura 2.4. Entonces, si el Hash que identifica al bloque cambia, ya no se mantiene la relación con el bloque anterior y el bloque posterior y por dicho motivo, ya se puede asegurar que ese bloque ha sido modificado y por tanto contiene información fraudulenta.





BLOQUE ORIGINAL AÑADIDO A LA CADENA



MISMO BLOQUE MODIFICANDO UNA TRANSACCIÓN

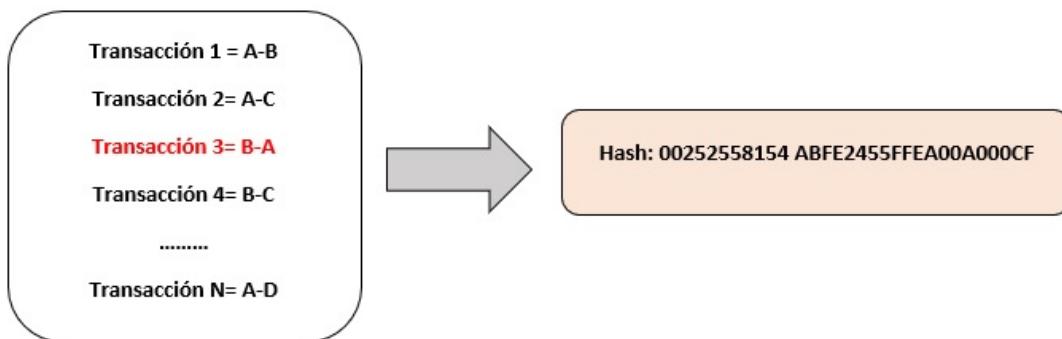
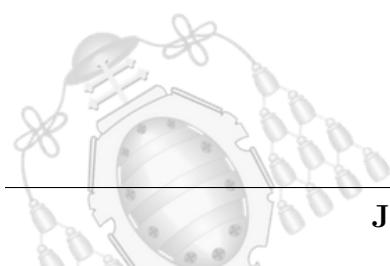


Figura 2.4.- Esquema de intento de modificación de un bloque

2.2.3.- Generación de nuevos bloques

En este apartado se describe cómo se realiza el acuerdo entre los nodos que forman la red para poder añadir nuevos bloques a la cadena. Anteriormente, se ha comentado que uno de los principios y características fundamentales de las redes de blockchain son los mecanismos de consenso que implementan, y que son utilizados por los nodos para llegar a un acuerdo a la hora de tomar decisiones cuando tienen que añadir un nuevo bloque a la cadena. Además, se han mencionado los principales tipos de mecanismos de consenso que se utilizan en las redes de blockchain. En este apartado se describe en profundidad el funcionamiento de los más importantes y los que tienen mayor relevancia a la hora del desarrollo de este proyecto.





2.2.3.1.- PoW - Proof of Work

Este algoritmo de consenso fue el primero que se usó en las redes de blockchain, pero en la actualidad sigue siendo de los más utilizados. En las redes en las que se utiliza este algoritmo de consenso, los nodos encargados de tomar las decisiones (validadores) también se conocen como mineros. En este caso, estos nodos mineros compiten entre ellos por generar el nuevo bloque, ya que el nodo (minero) que consiga terminar el bloque primero recibe una recompensa económica [6].

El procedimiento que deben cumplir los nodos mineros para poder añadir un nuevo bloque consiste en resolver un problema matemático. Este problema matemático está relacionado con el Hash del bloque. Los mineros tienen que conseguir obtener el valor exacto del Hash del nuevo bloque. Para conseguir esto se necesita una gran potencia computacional, ya que como se ha dicho antes, el Hash está compuesto por 64 símbolos en lenguaje hexadecimal, lo que hace que haya 64^{16} combinaciones diferentes [7]. Una vez que se consigue este Hash, antes de que se añada el nuevo bloque a la cadena, debe ser validado por al menos el 51 % de los nodos de la red. Esto permite evitar que un minero añada a la red un bloque fraudulento. Ya que de esta forma, para añadir nodos fraudulentos a la red hay que disponer del control del 51 % (como mínimo) de nodos de la red [8].

Dentro de este algoritmo de consenso es importante que se controle el nivel de dificultad para la resolución del problema matemático. En caso de que la complejidad sea muy elevada, conseguir minar un bloque requiere muchos recursos computacionales y por tanto un gasto energético muy alto. De lo contrario, si la dificultad del problema es muy baja, eso genera una inseguridad de la red, ya que la convierte en una red más vulnerable ante ataques [6].

2.2.3.2.- PoA - Proof of Authority

Este segundo algoritmo de consenso está basado en utilizar la identidad real de los nodos validadores como seguro de confianza y muestra de transparencia para que el resto de nodos confíen en él y poder encargarse de la generación de los nuevos bloques



que se añaden a la cadena de bloques. En este caso el número de nodos validadores es limitado y se eligen arbitrariamente dentro del grupo de nodos validadores confiables. El hecho de tener un número de nodos validadores limitado, convierte a este algoritmo de consenso en una opción muy útil en aquellos escenarios en los que prime la velocidad de generación de bloques [9].

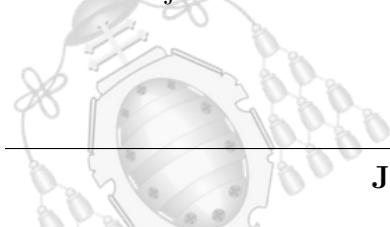
El funcionamiento de este tipo de algoritmo de consenso se basa en elegir los nodos que actúan como validadores confiables. Para realizar esta elección de los nodos se hace gracias a una votación realizada por un conjunto de nodos anteriormente autorizados. Gracias a este mecanismo de funcionamiento, se evita que un nodo malicioso pueda hacerse pasar por un validador de confianza y sabotear el funcionamiento de la red [9].

Como el algoritmo de consenso se basa en la reputación que tengan los nodos validadores dentro de la red, es muy importante que estos cuiden su identidad, de forma que sean un símbolo de confianza y transparencia para el resto de nodos. Además, aquellos nodos que son validadores en las redes de blockchain que usan el algoritmo de consenso de PoA son siempre los principales responsables en caso de que falle la red.

Como principal ventaja de este algoritmo de consenso destaca que el coste computacional es mucho más bajo que en el caso de el algoritmo de consenso PoW. Por dicha razón, el consumo energético es considerablemente inferior y por tanto se considera este tipo de algoritmo de consenso más amigable desde el punto de vista del medio ambiente. Como principal desventaja de este método se encuentra la pérdida del concepto de descentralización, ya que ahora el poder de toma de decisiones va a estar concentrado en unos nodos en concretos en lugar de estar distribuido a todos los nodos de la red como pasa en PoW [10].

2.2.3.3.- PoS - Proof of Stake

Este algoritmo de consenso, en castellano conocido como prueba de participación, se considera como una mejora del algoritmo PoW ya que aporta mejor seguridad y también mejor escalabilidad en las redes en las que se usa.





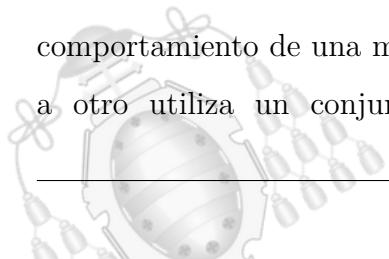
Al igual que en el algoritmo de PoW el objetivo principal de PoS es conseguir validar bloques para añadirlos a la cadena. Sin embargo, existe una gran diferencia entre ambos algoritmos. En PoW los nodos compiten, utilizando su capacidad computacional, para ver quién consigue resolver el problema matemático lo antes posible, mientras que en PoS se llega a un acuerdo a ver qué nodo validador es el elegido como encargado de crear y validar nuevos bloques. Estos bloques se eligen de manera aleatoria, pero la probabilidad de que alguno pueda ser elegido depende de ciertos parámetros que varían según la red. Uno de los más utilizados es el número de criptomonedas de esa red que tenga ese nodo. A mayor cantidad de monedas, mayor probabilidad de salir elegido como nodo validador [11].

La garantía de seguridad de las redes basadas en el algoritmo de consenso PoS se basa en la idea de que para poder llevar a cabo un ataque exitoso a la red, se debe poseer gran parte de las criptomonedas de esa red (para tener mayor probabilidad de ser el nodo elegido), y a ningún nodo legítimo de la red le va a resultar rentable realizar un ataque, porque se perjudicaría así mismo debido a la bajada de precio de esas criptomonedas y a la pérdida económica que eso supondría.

Este algoritmo de consenso tiene como principales ventajas que reduce considerablemente el gasto computacional en comparación con PoW, por lo que, al igual que PoA es mucho más favorable para el medio ambiente. Además, mantiene a la perfección la idea de sistema descentralizado, ya que todos los nodos pueden participar en la red. Otro de los aspectos favorables que presenta este algoritmo de consenso es que permite una mayor escalabilidad, ya que al no requerir una gran capacidad computacional para validar transacciones, puede usarse en redes que necesiten gestionar gran cantidad de transacciones por segundo [12].

2.2.3.4.- Istanbul Byzantine Fault Tolerant (IBFT)

Este algoritmo de consenso difiere considerablemente de los comentados anteriormente. En este caso, el funcionamiento de este algoritmo está basado en el comportamiento de una máquina de estados. Este algoritmo, para pasar de un estado a otro utiliza un conjunto de mensajes que sirven de información al resto de



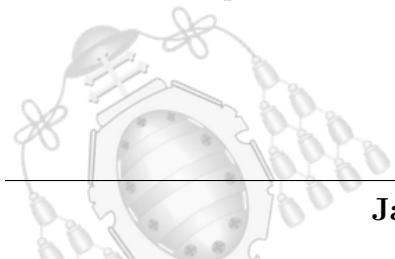


validadores de la red. Sin embargo, antes de detallar el funcionamiento, es necesario aportar una serie de definiciones que son imprescindibles para comprender mejor la explicación posterior. Estas definiciones son [13]:

- Validador: Nodo participante en la validación de un bloque
- Proponente: Nodo participante que es elegido para proponer un nuevo bloque en una ronda de consenso.
- Ronda de consenso: Empieza con un proponente creando una nueva propuesta de bloque y termina con la realización de un bloque.

Una vez comentados esos conceptos fundamentales, se pasa a describir el funcionamiento de este algoritmo de consenso. Para entender mejor este proceso, se describe paso a paso:

1. Inicialmente, los validadores escogen un nodo proponente utilizando la técnica de Round Robin [13]. El nodo elegido se encarga de proponer un nuevo bloque y lo anuncia al resto mediante un mensaje broadcast. Este mensaje recibe el nombre de 'Pre-Preparado'. Este estado inicial es conocido como Nueva Ronda.
2. Una vez que el resto de nodos han recibido la propuesta de nuevo bloque, se encargan de validar que sea adecuada. En caso afirmativo, se pasa al estado Pre-Preparado. Además, se realiza una transmisión broadcast del mensaje 'Preparado'.
3. Cuando los validadores han recibido dos terceras partes de mensajes 'Preparado' de los otros validadores, el validador elegido inicialmente pasa al estado Preparado y transmite mediante un mensaje broadcast el mensaje 'Commit'. Esto se realiza para informar al resto de validadores de que el nuevo bloque está preparado y se va a añadir a la cadena de bloques.
4. Posteriormente, los validadores esperan a recibir dos terceras partes de los mensajes 'Commit' para pasar al estado Committed y posteriormente añadir el nuevo bloque a la cadena.





5. Finalmente, una vez que el nuevo bloque se ha añadido a la cadena, se pasa al estado Final committed. Esto significa que los validadores están preparados para una nueva ronda de consenso.

Este proceso se refleja en la figura 2.5. En ella se observan los cinco estados diferentes que se han comentado, así como los mensajes o acciones utilizadas entre cada uno de ellos.

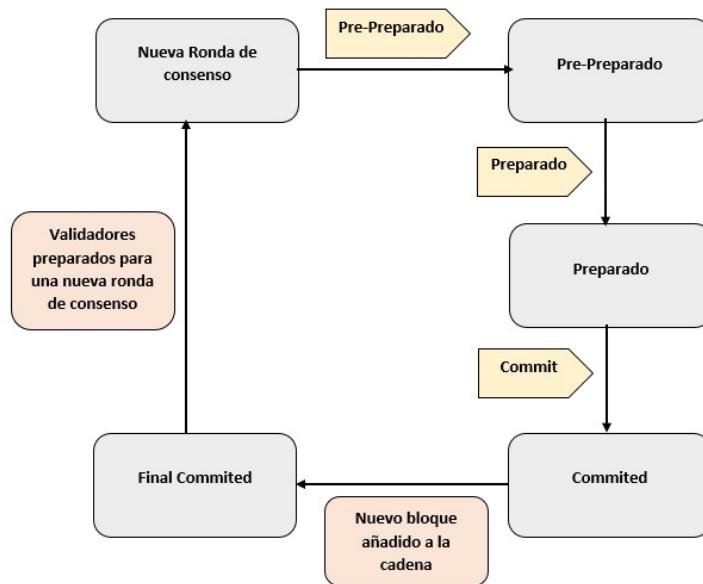
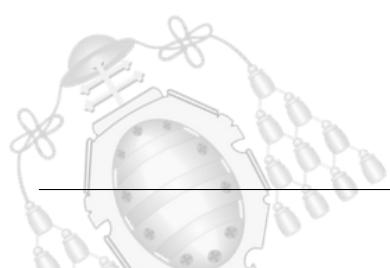


Figura 2.5.- Diagrama de estados del algoritmo de consenso IBFT

2.3.- Conceptos relacionados con la tecnología blockchain

Para complementar la información descrita en el apartado anterior respecto al funcionamiento de la tecnología de blockchain, en este apartado se explican algunos conceptos que están ligados directamente con el funcionamiento de esta tecnología y que son importantes para entender todo lo que engloba a blockchain. Además, estos conceptos también son importantes de cara al desarrollo del proyecto y a los capítulos siguientes del presente documento.





2.3.1.- Dificultad

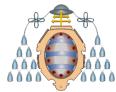
El término de dificultad generalmente es utilizado en aquellas redes de blockchain que utilizan el algoritmo de consenso de Proof of Work (PoW), aunque puede ser utilizado junto al resto de algoritmos de consenso ya comentados en este documento. Este término indica la complejidad que tiene una red de blockchain para generar nuevos bloques. Este parámetro es regulado automáticamente cada cierto tiempo. Por ejemplo, en la red de blockchain que da soporte a las Bitcoin, el nivel del parámetro es regulado cada 2016 bloques de la cadena [14].

La principal función de este parámetro es regular la velocidad y la complejidad a la que se generan nuevos bloques. Si una red de blockchain tiene una dificultad muy elevada, los bloques tardan más en generarse. También, al ser más difícil resolver el problema matemático (PoW) se necesita mayor capacidad computacional lo que supone que menos nodos dispongan de los medios necesarios para conseguirlo. Además, el aumento de la dificultad también implica un mayor gasto energético. Pero a su vez, el aumento de la dificultad supone un aumento de la seguridad de la red, ya que como se ha comentado, menos nodos disponen de las capacidades computacionales para generar nuevos bloques y por tanto es más difícil, para un nodo malicioso, tomar el control de la red.

Por otro lado, si se habla de una dificultad baja, el número de bloques creados aumenta ya que se necesita menos tiempo para poder resolver el problema matemático. Además, al ser más sencillo, no requiere de una capacidad computacional y por dicho motivo muchos más nodos pueden optar a resolver dicho problema. Esto reduce el nivel de seguridad de la red, ya que resulta más sencillo para un nodo malicioso poder tomar el control de la red.

2.3.2.- Recompensas

Las recompensas dentro de las redes de blockchain son un elemento clave para preservar el buen funcionamiento de la red y motivar a los nodos validadores o mineros para que sigan realizando esas labores. Estas son otorgadas cada vez que se completa un



bloque y se añade a la cadena. Generalmente las recompensas son tokens de la propia red de blockchain. En estos casos, cuando se habla de tokens se hace referencia a la unidad económica utilizada para cuantificar todos los gastos, transacciones económicas y elementos similares que influyen en la red. Un claro ejemplo de estos tokens son las criptomonedas [15]. Por ejemplo, en la red de blockchain que da soporte a Bitcoin. Los nodos mineros reciben una recompensa en Bitcoin cada vez que se añade un nuevo bloque. Esto también ocurre en la red de blockchain Ethereum (la cuál es uno de los temas principales de este proyecto). Los nodos validadores de la red Ethereum, reciben una recompensa en Ethers cada vez que consiguen añadir un nuevo bloque a la cadena. Pero este caso de la red de Ethereum, la recompensa no es una cuantía fija, sino que depende de un valor denominado Gas (se explica en capítulos posteriores).

Estas recompensas tienen gran importancia en el funcionamiento de las redes de blockchain ya que aportan motivación a los nodos encargados de hacer que la red funcione correctamente. En caso de no existir este tipo de recompensas, no sería rentable para los nodos encargados de crear nuevos bloques realizar ese proceso, ya que generar nuevos bloques supone un gasto elevado en energía eléctrica así como en hardware. Y en caso de que no hubiese nodos encargados de validar los nuevos bloques, la red no sería segura y por tanto la tecnología blockchain no tendría utilidad [15].

2.3.3.- Smart Contracts

En castellano conocidos como Contratos inteligentes, son un elemento clave que permite darle utilidad a las redes de blockchain. Se puede definir estos contratos inteligentes como una serie de instrucciones que se almacenan en la red de blockchain y que se pueden ejecutar automáticamente cuando se cumplen las condiciones programadas en ellos [16].

Los contratos inteligentes son scripts programados, generalmente utilizando el lenguaje de programación Solidity, que están almacenados en la red de blockchain y que se ejecutan automáticamente de forma transparente e inmutable debido a que están almacenados en la cadena de bloques. El uso de smart contracts favorece a que se ejecuten acciones sin necesidad de tener una entidad intermediaria que gestione todo



el proceso. Mediante estos contratos inteligentes se pueden programar aplicaciones que utilizan la red de blockchain como plataforma (Dapps).

Los contratos inteligentes aportan muchos beneficios a la hora de rentabilizar el uso de los cadenas de bloque, pero siempre que se usa un contrato inteligente, se debe tener en cuenta que si el contrato tiene algún fallo a la hora de su programación, eso puede suponer grandes riesgos en la seguridad. Además, que una vez que un contrato es registrado en la cadena de bloques, ya no se puede modificar su funcionamiento [17].

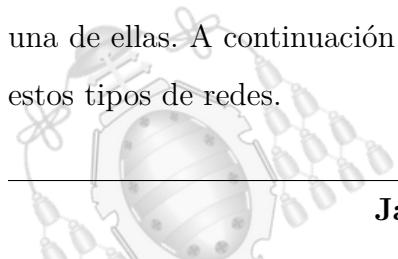
2.3.4.- Dapp - Aplicaciones distribuidas

Este tipo de aplicaciones son las que se ejecutan sobre una red distribuida y no requieren de una entidad central (como puede ser un servidor) para ejecutarse [18]. En las aplicaciones tradicionales, toda la información de los usuarios está ligada a un punto central desde el que se controla toda la red, en cambio, en este nuevo paradigma de aplicaciones, los nodos que conforman la red son los encargados de que la aplicación funcione correctamente.

La idea principal de las aplicaciones descentralizadas es que estén implementadas sobre una red de blockchain y que utilicen smart contracts para poder ejecutar todas las acciones que deben llevar a cabo para que las aplicaciones cumplan su cometido. De esta manera, las aplicaciones descentralizadas pueden aprovecharse de las mejores características de las redes de blockchain, por ejemplo, la seguridad e inmutabilidad de la información utilizada así como la descentralización de la información, de forma que, se evitan puntos únicos de fallo y vulnerabilidades explotables por ataques informáticos.

2.4.- Tipos de redes de blockchain

Cuando se explica el funcionamiento de blockchain, también es importante mencionar los diferentes tipos de redes de blockchain que se pueden encontrar en la actualidad. Esta clasificación se basa en el ámbito para el que está pensado usar cada una de ellas. A continuación se explican las principales características de cada uno de estos tipos de redes.





2.4.1.- Redes blockchain pública

Este es el primer tipo de red de blockchain desplegada. La idea de una red de blockchain pública es aquella que está pensada para que pueda ser accesible en Internet. Por tanto, todo el mundo puede contribuir al funcionamiento de la misma. En este tipo de redes, son los propios nodos los que deciden qué bloques son agregados a la red [19]. Alguna de las redes públicas más importantes a día de hoy son Bitcoin y Ethereum.

La principal ventaja de este tipo de redes es que cumplen totalmente una de las bases de la tecnología blockchain como es la descentralización y la eliminación de cualquier entidad central encargada de gestionar toda la información de la red.

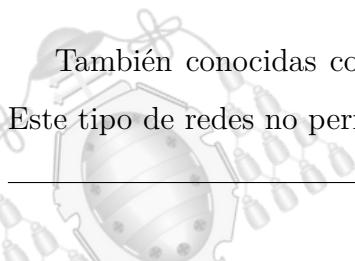
2.4.2.- Redes blockchain privadas

Una vez que el blockchain se ha vuelto popular, algunas empresas han empezado a crear sus propias redes de blockchain para aplicarlas a sus servicios. Este tipo de red consiste en redes de blockchain que son gestionadas por las propias empresas y que solo tienen acceso a ellas los miembros que son autorizados por las entidades encargadas de controlar la red. Desde el punto de vista de una empresa, este tipo de red le aporta una mayor seguridad a la información almacenada en la cadena de bloques, ya que ellos mismos son los encargados de autorizar, o no, a los diferentes usuarios de la red.

La principal desventaja de este tipo de redes es que se alejan de la idea fundamental de la tecnología de blockchain, la descentralización de la red. Como se dijo anteriormente, estas redes sí necesitan una entidad central encargada de mantener la red y dar acceso a los usuarios que se quieren conectar a la misma.

A día de hoy, las redes privadas de blockchain más importantes son Corda o Hyperledger [19].

2.4.3.- Redes de blockchain permisionadas



También conocidas como redes de consorcio, son un tipo de redes semi-públicas. Este tipo de redes no permiten que cualquier nodo se encargue de la validación de los



nuevos bloques que se añaden a la cadena. En este tipo de redes hay una serie de nodos, pre-seleccionados y que pertenecen a las empresas que forman parte del consorcio, que son los encargados de realizar la validación de los bloques [20].

La principal diferencia de este tipo de redes con las redes privadas, es que en las redes privadas el control de la red está otorgado a una única entidad. En cambio, en las redes permisionadas el control está otorgado a todo el grupo de empresas u organizaciones que forman parte del consorcio que despliega la red.

Uno de los grandes beneficios de este tipo de redes es que la responsabilidad de su mantenimiento corresponde a varias entidades simultáneamente, lo que garantiza que el funcionamiento de la red esté mucho más controlado. Además, al tratarse de un consorcio de varias entidades, permite la combinación de entidades de sectores diferentes trabajando sobre una misma red de blockchain. Un claro ejemplo de este tipo de redes es el consorcio Alastria, el cual es una parte fundamental de este proyecto y que se explica en capítulos posteriores.

2.5.- Principales sectores de aplicación de blockchain

Desde su origen, la tecnología blockchain se relaciona en la mayoría de casos directamente con las criptomonedas, siendo el Bitcoin su mayor exponente. Siempre que se habla de tecnología blockchain es inevitable que aparezca la palabra Bitcoin detrás. Pero esto es un error, ya que la tecnología blockchain puede utilizarse en muchos más sectores a parte de las criptomonedas.

El sector económico y financiero es uno de los sectores que más ha invertido en aplicar la tecnología de blockchain a sus actividades con el fin de poder mejorar la seguridad y la eficiencia de algunas de sus tareas básicas. Un término que está muy ligado a la aplicación de blockchain al sector financiero es Fintech. Este término, se refiere a la mejora de este sector utilizando los nuevos avances tecnológicos, en particular blockchain [21]. La tecnología de blockchain puede aportar a este sector seguridad en el intercambio de dinero, así como un registro inmutable de todas las transacciones que se realizan. Además, gracias a los beneficios que ofrece respecto a la identificación



de identidad de los usuarios también permite evitar cualquier tipo de fraude con las transacciones.

Otro de los sectores en lo que se cree que el blockchain pueda tener gran influencia en el futuro es el sector sanitario. En este caso, si se despliega una red de blockchain particular para todos los centros hospitalarios de un país, el historial médico de los pacientes estaría almacenado de una forma segura pero accesible para todo aquel personal sanitario autorizado y sería común para todos los centros, en lugar de tener un almacenamiento más limitado por regiones.

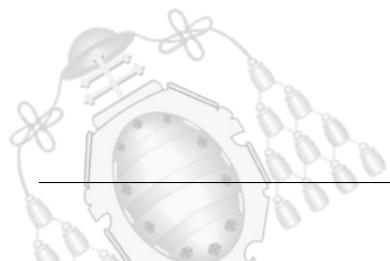
Por otra parte, el sector de los seguros y del registro de propiedades también pueden ser otros de los grandes beneficiados con la incorporación de la tecnología de blockchain. Ambos sectores pueden aprovechar los beneficios de inmutabilidad y seguridad de la información que blockchain ofrece para poder evitar falsificación de documentos.

2.6.- Inconvenientes de la tecnología de blockchain

Los principales inconvenientes de la tecnología de blockchain en la actualidad, no están ligados a sus características técnicas, si no que se refieren a su adopción en la sociedad. Tanto desde un punto de vista legal como social.

El principal inconveniente que presenta la tecnología de blockchain es la falta de regulación legal existente. Es decir, la mayoría de países del mundo aún carecen de una regulación legal extensa para aplicar la tecnología de blockchain a los servicios citados anteriormente. De esta forma, se limita mucho el campo de aplicación de la tecnología de blockchain ya que no tiene una legislación que le respalde.

Otro de los grandes problemas que presenta esta tecnología es el desconocimiento generalizado en la sociedad. Esto hace que se vea esta tecnología como una tecnología poco útil y con pocos campos de aplicación. Además, debido a la falta de información también abunda la falta de confianza en ella.





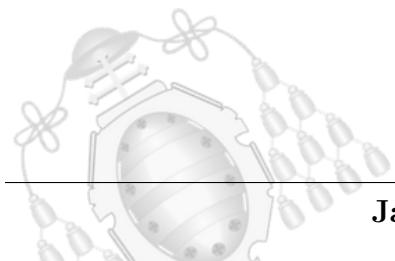
3. Hipótesis de partida y Alcance del proyecto

En este capítulo se definen los pilares básicos en los que se fundamentan las investigaciones realizadas en este proyecto. Por dicha razón, se considera este capítulo esencial para poder entender cuál es el principal propósito del proyecto así como entender todo lo analizado durante el transcurso de este.

3.1.- Ethereum

Como primer elemento clave de este proyecto se encuentra la red pública de blockchain Ethereum. Esta red tiene su origen oficial en 2015 y fue creada por Vitalik Buterin [22]. Se trata de la red de blockchain pública de carácter general más importante. Cuando se dice que es de carácter general, se hace referencia a que no es una red destinada a un propósito en concreto (como por ejemplo la red de Bitcoin) sino que es una red pensada para que sobre ella se puedan implementar multitud de aplicaciones descentralizadas y pueda ser usada en múltiples sectores.

La red de Ethereum cumple con todas las características principales de la tecnología blockchain. Es una red totalmente descentralizada en la que no existe ninguna entidad central encargada de gestionar la red, sino que son los propios nodos los encargados de verificar que el funcionamiento de la red sea correcto. Además, también mantiene la seguridad en la información y la inmutabilidad de los datos, lo que la convierte en una excelente plataforma de desarrollo para Dapps que pretenden trabajar con datos o material sensible. En la actualidad, las aplicaciones más importantes que se han desarrollado sobre esta plataforma están relacionadas con crear carteras de criptomonedas, aplicaciones de intercambio de activos, juegos y apuestas online [23].





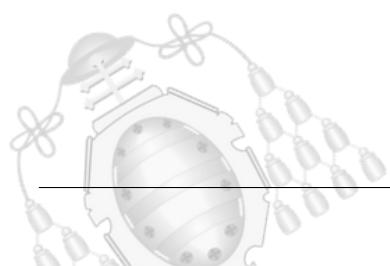
3.1.1.- Criptomoneda de Ethereum

Otro elemento que es indispensable para el correcto funcionamiento de la red de blockchain Ethereum y que se debe considerar, es la criptomoneda nativa de esta red de blockchain. Esta se conoce como Ether (ETH) o Ethereum (puede generar confusión al presentar el mismo nombre que la red) [23]. Esta criptomoneda es un activo digital que dentro de la cadena de bloques, puede tener muchos usos diferentes. Por ejemplo, esta criptomoneda puede utilizarse como activo de inversión, al igual que otras monedas como el Bitcoin, su precio oscila muy frecuentemente lo que le convierte en un activo atractivo dentro del mundo de las inversiones.

Sin embargo, lo más importante de esta moneda es que es el combustible que mueve la red [24]. Los Ethers son usados como forma de pago para las recompensas de los nodos validadores encargados de añadir nuevos bloques a la cadena. Como se ha comentado en el capítulo anterior, en las redes públicas, en las que no hay ninguna entidad central encargada del gobierno de toda la red, los propios nodos son los encargados de garantizar el funcionamiento de la red y la seguridad de la misma. Para esto se utilizan, los ya citados, algoritmos de consenso. En la red de Ethereum se utiliza el algoritmo de Proof of Work (PoW). Este genera un gasto energético elevado y requiere de un hardware complejo. Todos estos motivos hacen que sea necesario repartir algún tipo de recompensa para los nodos validadores de bloques, ya que de lo contrario, no sería rentable realizar todas esas operaciones y si ellos no las realizasen, la red de Ethereum no funcionaría. Estas recompensas, son otorgadas mediante Ethers.

3.1.2.- Gas en la red Ethereum

Siguiendo con la descripción de los elementos y características más importantes de la red Ethereum, es indispensable comentar a qué se refiere el término Gas dentro de la red blockchain y la importancia que tiene para el funcionamiento de la red.





3.1.2.1.- Qué es el Gas

El Gas se considera como el coste asociado que tiene realizar una operación o un conjunto de ellas sobre la plataforma Ethereum [25]. El término operación en este caso engloba a diversas acciones como pueden ser realizar una transacción, ejecutar un smart contract o crear una aplicación descentralizada. Se debe considerar que cada tipo de acción tiene un coste asociado, es decir, según la complejidad y el requerimiento de recursos de dicha acción el coste en Gas es más o menos elevado. Por ejemplo, la ejecución de un smart contract que consta de dos instrucciones requiere menos recursos que la ejecución de un smart contract que consta de quince instrucciones. Por dicho motivo, la ejecución del primer smart contract tiene un coste de Gas menor que la ejecución del segundo.

3.1.2.2.- Utilidades principales del Gas

El Gas es un elemento fundamental para mantener el correcto funcionamiento de la red Ethereum ya que tiene varias utilidades dentro de la red que permiten preservar su funcionamiento. Algunas de estas utilidades son las siguientes [25]:

- **Cuantificar el coste de cada acción:** Esto hace relación a lo explicado en el párrafo anterior.
- **Mantener la seguridad de la red:** Está relacionado con el hecho de que se asocie a cada acción en la red un gasto. De esta forma se evita ejecutar acciones inservibles y que colapsen la red. Además, el hecho de tener un gasto asociado a cada acción, limita las intenciones de cualquier atacante de intentar realizar un ataque de denegación de servicio basado en ejecutar cientos de miles de transacciones en muy poco tiempo, ya que ejecutar tantas acciones supondría un gasto considerable.
- **Sirve de recompensa para los mineros:** El Gas pagado cada vez que se realiza una acción sobre la red de Ethereum sirve para pagar parte de la recompensa que reciben los nodos validadores (mineros) cada vez que añaden un nuevo bloque a la cadena.





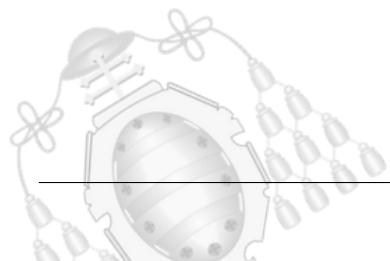
3.1.2.3.- Uso del Gas como recompensa

Como se comentó en el apartado anterior, parte del Gas pagado a la hora de realizar cada acción sobre la red de Ethereum es utilizado para pagar las recompensas que reciben los mineros cada vez que añaden un nuevo bloque a la cadena. Esto es una parte esencial del buen funcionamiento de Ethereum ya que permite mantener el equilibrio entre el esfuerzo que realizan los mineros por preservar el orden de la red y la recompensa que reciben por ello. En este subapartado se comenta cómo se realiza este proceso.

Lo primero que se debe tener en cuenta es que el Gas no es una unidad con la que se pueda pagar directamente, sino que solo sirve para cuantificar el gasto asociado a la ejecución de cada acción. Por eso, se necesita una equivalencia entre el Gas y el Ether. Esta equivalencia permite abonar en Ethers la recompensa a los mineros en función del coste en Gas de cada acción. Es necesario destacar la importancia de la equivalencia entre Gas y Ethers a la hora de realizar los pagos, ya que el Ether es una unidad muy volátil, por lo que su valor cambia constantemente, mientras que el Gas es un valor que permanece constante ya que es un parámetro definido en el protocolo de la red Ethereum. Esto permite que el coste de las transacciones sea siempre el mismo y no haya que modificar el protocolo de la red cada vez que haya una variación del precio del Ether [25].

3.1.3.- Inconvenientes de Ethereum

El principal inconveniente de la red de Ethereum está ligado con el Gas que hay que pagar cada vez que se ejecuta una transacción. Desde el punto de vista de cualquier empresa, tener que pagar cierta cantidad por cada acción que debe ejecutar hace que no le resulte rentable implementar sus servicios sobre la plataforma Ethereum. Por dicho motivo, desde una perspectiva empresarial, la red de Ethereum no es la mejor opción para utilizar la tecnología de blockchain en la actividad empresarial.





3.2.- Quorum

En esta nueva sección del documento se plantea una alternativa a Ethereum, la cual está más orientada al mundo empresarial. Se trata de la red de blockchain Quorum. Esta red fue creada a partir de una variante de Ethereum, por lo que comparte con ella la mayoría de características [26]. Se trata de una red de blockchain permisionada que permite la implementación y uso de smarts contracts de manera privada.

El concepto de variante en esta situación se refiere a una bifurcación de la cadena de bloques principal, en este caso de la cadena de bloques Ethereum. Este tipo de bifurcaciones se realizan debido a modificaciones en el código base de la cadena o a cambios de configuración que provocan que la cadena se oriente hacia otros ámbitos.

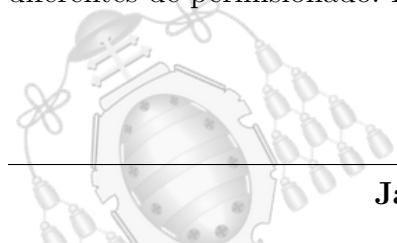
3.2.1.- Características de Quorum

El origen de Quorum reside en la necesidad de adaptar la red de Ethereum a un entorno empresarial donde las empresas que utilizan este tipo de tecnología no tienen que pagar una tasa por cada acción que realizan. Esto hace que la red de Quorum no necesite una criptomoneda nativa como son los Ether en la red de Ethereum, ya que cada acción que se ejecuta en la red no lleva un coste asociado.

Además, otra característica que se añade respecto a la red de Ethereum, y que convierte a este tipo de red en una opción aún más interesante para el entorno empresarial, es el permisionado de la red. Esto significa que los nodos van a poder definir qué nodos se incorporan a la red y con cuáles están conectados. De tal forma que se puede proteger la red mediante este control de acceso a la misma. Así se puede evitar que nodos con intenciones maliciosas se añadan a la red.

3.2.1.1.- Tipos de permisionado en Quorum

Otra característica de las redes basadas en Quorum, es que pueden utilizar dos tipos diferentes de permisionado. Estos tipos son [27]:





- **Permisionado de red básico:** Controla qué nodos pueden conectarse con otro nodo dado.
- **Permisionado de red mejorado:** Este tipo de permisionado está basado en el uso de un smart contract que permite flexibilizar el permisionado de la red según las necesidades. De esta forma se puede variar la gestión de las conexiones de los nodos.

3.2.1.2.- Algoritmos de consenso en Quorum

En las redes de blockchain que se basan en Quorum, no se requiere el uso del algoritmo de consenso PoW como se utiliza en la red Ethereum. Este tipo de algoritmo de consenso no es adecuado para redes más enfocadas a la empresa, ya que no es rentable para estas utilizar un volumen tan grande de recursos para poder crear y añadir nuevos bloques en la cadena. Además, al evitar el mecanismo de PoW se consigue mayor velocidad en la red. Los principales algoritmos usados en las redes basadas en Quorum son:

- Algoritmo de consenso Raft.
- Algoritmo de consenso IBFT.

El primero de ellos es un algoritmo de consenso similar a Proof of Stake (PoS). Dicho algoritmo se conoce como Raft [28]. Este algoritmo sigue la misma idea que PoS, ya que para elegir qué nodos pueden tomar una decisión valora la participación que tienen los mismos en la red.

Este algoritmo de consenso está enfocado para usarse en redes permisionadas que tienen un propósito empresarial. Permite una mayor velocidad de ejecución de transacciones en la red. Cuando se usa este algoritmo de consenso en una red de blockchain, los nodos pueden tomar tres roles diferentes: líder, verificador o aprendiz. Los líderes son los encargados de generar nuevos bloques. Además, pueden gestionar qué nodos verificadores se añaden a la red y también qué nodos pueden cambiar su rol de líder a verificador. Los nodos verificadores son aquellos encargados de comprobar las decisiones de los líderes. También tienen potestad para añadir nuevos nodos



verificadores o aprendices. Los aprendices no pueden tomar decisiones ni tampoco comprobarlas, simplemente tienen acceso a la información de la red, pero sin tener ningún poder sobre ella [29].

En segundo lugar está el algoritmo IBFT. Este algoritmo ya se ha explicado en el apartado 2.2.3.4. En este caso, solo queda destacar las ventajas que tiene frente a otros algoritmos desde el punto de vista de que no requiere un coste computacional elevado ni tampoco tiene tanta complejidad de implementación como en el caso de otros algoritmos.

3.2.2.- Aplicación de Quorum

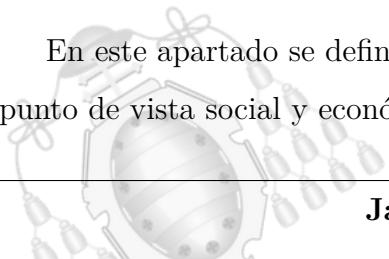
La tecnología de Quorum principalmente se aplica para el desarrollo de redes de blockchain de carácter empresarial. En la actualidad, uno de sus mayores exponentes es la red del consorcio Alastria. La red utilizada en Alastria permite englobar los servicios de múltiples empresas que pertenecen a sectores diferentes en una misma red. Este tipo de aplicaciones de la tecnología de blockchain hace que aumente su popularidad en la sociedad actual y que, por tanto, cada vez más empresas confíen en esta tecnología como plataforma de desarrollo para su actividad empresarial. Por dicho motivo, el objetivo principal de este proyecto se basa en la investigación de uno los planes que tiene el consorcio Alastria en cuanto a la mejora de las redes de blockchain que utiliza.

3.3.- Alastria

En esta sección se describe otro de los pilares básicos de este proyecto, el consorcio de Alastria. Es una parte esencial de este trabajo, ya que uno de los principales objetivos es investigar la viabilidad y dificultad de uno de los propósitos que tiene Alastria de cara al futuro para mejorar su funcionamiento y sus servicios.

3.3.1.- Características generales

En este apartado se definen las características más relevantes de Alastria desde un punto de vista social y económico.





3.3.1.1.- Origen de Alastria

Alastria tiene su origen en el año 2017 y fue creado inicialmente bajo el nombre de Red Lyra. Se trata de la primera asociación sin ánimo de lucro creada en España y que basa su funcionamiento en la tecnología de blockchain. La idea fundamental de esta asociación es democratizar e incentivar el acceso a la tecnología blockchain por parte de empresas de múltiples sectores, de forma que todas puedan convivir y obtener los mejores beneficios de la tecnología usando una misma red de blockchain para ofrecer sus servicios a sus clientes [31]. Algunas de las empresas más importantes que tomaron parte en la creación de esta asociación fueron: Banco Santander, Caja rural, Endesa, Gas Natural Fenosa, Everis o Iberdrola entre otros [32].

3.3.1.2.- Alastria en la actualidad

En la actualidad, Alastria ha llegado a ser un proyecto de referencia en el desarrollo de la economía digital a nivel nacional ya que promueve métodos innovadores para el despliegue de diferentes tipos de servicios utilizando una red de blockchain como plataforma de desarrollo. Este planteamiento le ha permitido convertirse en una asociación atractiva para muchas entidades, que ven en ella una oportunidad de aprovechar los beneficios del blockchain, sin necesidad de realizar grandes inversiones para el despliegue de su propia red. Algunas de estas empresas son: Indra, Mapfre, Repsol, Telefónica, etc. Además, muchas universidades y parques tecnológicos se han unido a esta asociación con el fin de aprovechar los beneficios de la tecnología de blockchain [33].

3.3.1.3.- Objetivo de Alastria

Uno de los principales objetivos que plantea Alastria es convertirse en un referente a nivel de evolución y desarrollo de la economía digital. Pretende ser el ejemplo a seguir por muchas organizaciones que quieran mejorar sus servicios aprovechando los servicios de las redes de blockchain. Con esto, también pretende fomentar el uso del blockchain en un ámbito multisectorial de forma que el acceso y adopción de esta tecnología resulte



mucho más sencillo para las empresas, independientemente del sector del que formen parte.

En la figura 3.1 se plantea una representación más visual de la misión y la visión que tiene Alastria. Es decir, de lo que pretende ser en la actualidad en cuanto a innovación tecnológica y económica, y también lo que espera llegar a ser en un futuro [33].

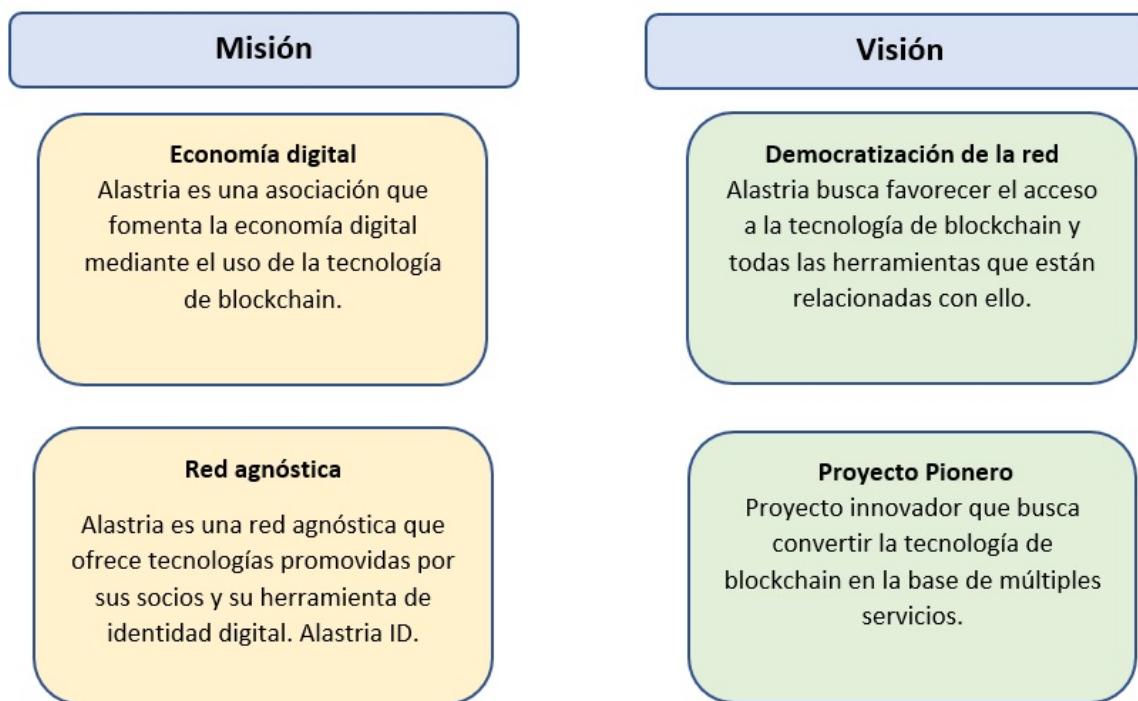


Figura 3.1.- Esquema con la misión y visión de Alastria

3.3.1.4.- Ejemplos de casos de uso

La plataforma de blockchain desarrollada por el consorcio de Alastria sirve como base para decenas de aplicaciones que basan su funcionamiento en las características de la tecnología de blockchain. Estas aplicaciones pertenecen a múltiples sectores, lo que pone en evidencia que se está cumpliendo uno de los objetivos iniciales de la red de Alastria, dar soporte a empresas de diversos sectores con una plataforma de blockchain. Alguna de estas aplicaciones son las siguientes [33]:

- Clockchain: Aplicación que permite a las empresas mantener un registro horario de los trabajadores acuerdo a la norma vigente de protección de datos.



- Certification of Wine: Aplicación que permite obtener la trazabilidad del vino. Desde el origen de la uva utilizada, pasando por todas las etapas, hasta su venta final.
- SmartDegrees: Aplicación que permite la validación de títulos y certificados académicos, de forma que se puedan autenticar y añadir al currículum de manera oficial.

Se han elegido estas aplicaciones como muestra de todas las que están desarrolladas en la plataforma de Alastria. Cada una pertenece a un sector diferente y tiene una finalidad completamente distinta. Esto permite demostrar que la idea original de Alastria, basada adaptarse a multitud de ámbitos, se está cumpliendo.

3.3.2.- Características técnicas

En este apartado se describen las características de Alastria desde un punto de vista técnico, tratando de relacionar dichas explicaciones con lo detallado en el capítulo 2. Principalmente, se hace énfasis en el tipo de red que usa Alastria y las características de la misma que la hacen una opción adecuada para los propósitos que tiene Alastria.

3.3.2.1.- Tipo de red utilizada

Como ya se ha comentado anteriormente, Alastria es una asociación que congrega a una amplia gama de empresas y organizaciones. Por dicho motivo, la red de blockchain que utiliza tiene que estar pensada para desarrollar la actividad empresarial. Siguiendo esta idea, y basándose en lo explicado en el capítulo 2, se puede intuir que la red de blockchain principal utilizada por Alastria está basada en la tecnología de Quorum [34].

La red principal de Alastria es conocida como Red T. Dicha red está basada en una plataforma de blockchain como es Quorum. Es decir, una red de tipo permisionado, en la que no existe una criptomoneda propia de la red y las diferentes acciones que se ejecutan no requieren un gasto adicional al pagar por cada transacción. Todas estas características hacen que sea un escenario ideal para el despliegue de los servicios ofrecidos por las empresas que son socias de Alastria. En la figura 3.3 se muestra un



esquema que permite entender mejor el significado de permisionado y su posición en una escala que compara las redes totalmente descentralizadas con las redes distribuidas [33].

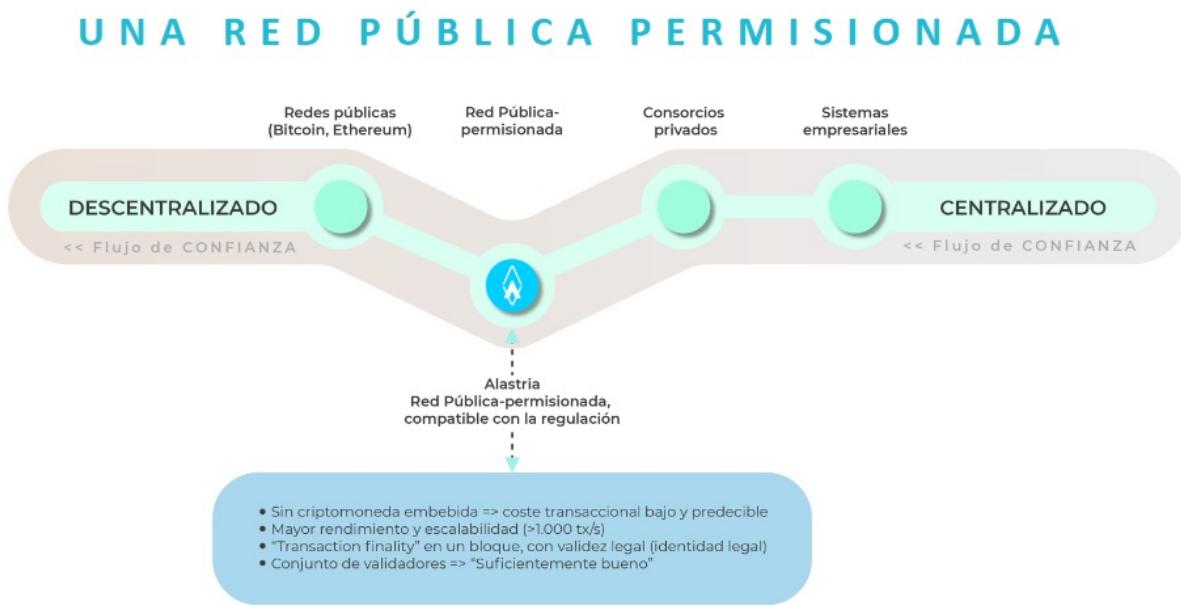


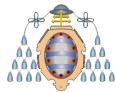
Figura 3.2.- Esquema de tipo de red

3.3.2.2.- Red T

La Red T de Alastria es la red principal que ha estado funcionando desde que la creación de esta asociación. La Red T está basada en la tecnología de Quorum. Esto significa, que es una red que admite permisionado. Por dicho motivo, todos los nodos tienen que estar autorizados para poder conectarse a la red. Además, este tipo de red no requiere el uso de una criptomoneda nativa de la red ya que para ejecutar cualquier tipo de acción sobre la red no es necesario pagar una determinada tasa. Esto también es una ventaja para que las empresas puedan desarrollar más eficazmente su labor, ya que les permite ahorrarse un gasto extra.

Otra característica que se debe comentar de esta red es que los nodos que la forman puede ser de tres tipos diferentes. Estos tipos son [34]:

- Nodos regulares: Son los encargados de tener copias de toda la información almacenada en la cadena de bloques. Además, aceptan las decisiones tomadas por los nodos validadores y ejecutan las transacciones en la red.



- Nodos Permisionadores: También son conocidos como bootnodes. Son los nodos encargados de localizar nuevos nodos para añadirlos a la red y permitir las conexiones a la red a través de ellos.
- Nodos validadores: Son los nodos encargados de generar y añadir nuevos bloques a la cadena y los encargados de ejecutar el algoritmo de consenso utilizado.

En la actualidad, existen 18 nodos validadores, 5 nodos permisionadores y 126 nodos regulares [35].

Sin embargo, a pesar de que esta red ha sido correctamente aceptada por todos los participantes en Alastria, se cree que es insuficiente para cumplir con los principios básicos de Alastria, ya que se define a Alastria como una red de blockchain agnóstica, es decir, que no confía su desarrollo a una única plataforma de blockchain [34]. Por ese motivo, en los últimos meses ha surgido la idea de desarrollar una nueva red, pero esta vez basada en otra tecnología diferente a Quorum. Esta red es la actualmente conocida como Red B.

3.3.2.3.- Red B

La Red B es la nueva red creada para la red Alastria. Inicialmente, esta red es definida como una red complementaria a la red T anteriormente comentada. El principal motivo para que esta red se haya puesto en marcha es evitar depender de una única plataforma de blockchain. Hasta el momento, toda la red de blockchain de Alastria estaba basada en la tecnología de Quorum. En cambio, ahora que la Red B ha empezado a funcionar, ya se presenta una alternativa a la red T original.

La Red B fue puesta en marcha el 4 de Febrero de 2020 [36]. Esta red inicialmente cuenta con solo 4 nodos. Se espera que durante este año y los próximos años se añadan nuevos nodos, ya que las mismas funciones que se desarrollan sobre la Red T, se pueden desarrollar sobre esta nueva red.

Por otra parte, también es importante recalcar las características generales más importantes de esta nueva red, ya que a parte de ser una gran evolución para Alastria,





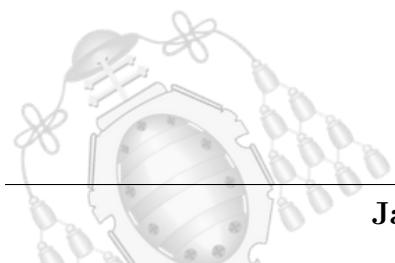
también es una de las piezas clave del desarrollo de este proyecto. Las características fundamentales de esta nueva red son [36]:

- Esta red está basada en la tecnología de Hyperledger Besu (se explica en el capítulo 4 en mayor profundidad).
- Esta red añade nuevas características de privacidad de la información almacenada.
- Permite regular el control de acceso con nuevas funciones de permisionado.
- Flexibiliza los requerimientos para la implementación de servicios empresariales sobre la plataforma.

3.3.3.- Objetivos futuros de Alastria

Una vez que la red B también está en funcionamiento, se empiezan a realizar los primeros pasos para alcanzar el objetivo final de la red de Alastria. Este objetivo consiste en lograr una multired híbrida, que englobe redes implementadas utilizando diferentes tecnologías, de tal forma que dichas redes no actúen como redes independientes, sino que las diferentes tecnologías de blockchain funcionen en una red homogénea para que se puedan aprovechar al máximo todas las ventajas de las diferentes tecnologías. Por tal motivo se pretende que en el futuro se pueda conseguir la interoperabilidad total entre la Red B y la Red T. Conseguir esto significaría un gran avance para la asociación Alastria, ya que se habría alcanzado uno de sus principales propósitos tecnológicos.

Otro de los objetivos de Alastria, es implementar una nueva red, la Red H, de forma que fuese otra nueva opción para incorporar una plataforma de blockchain como base a los servicios ofrecidos por las diferentes empresas que son parte de este consorcio. Esta nueva red H, también se pretende que pase a formar parte de la multired homogénea junto a las redes B y T ya comentadas anteriormente.



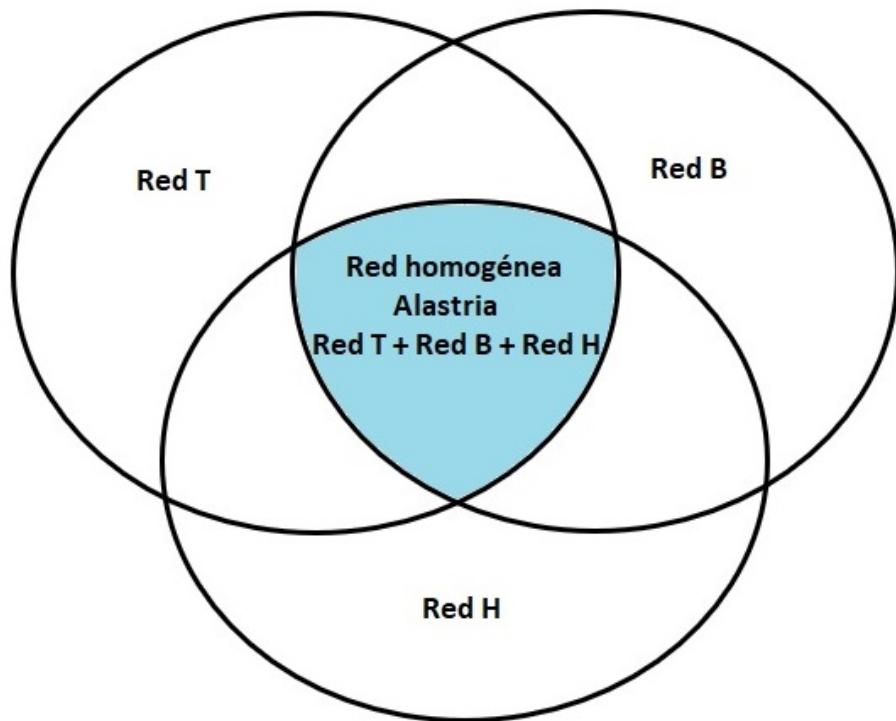
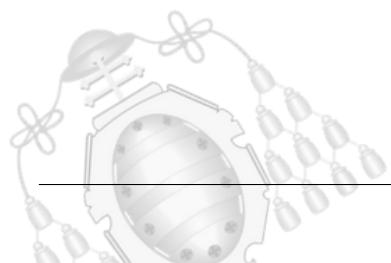


Figura 3.3.- Esquema de tipo de red

Este objetivo de formar una red que combine las diferentes tecnologías es algo clave para el desarrollo de Alastria. Por dicho motivo, el principal tema de investigación de este proyecto se alinea con el objetivo de Alastria, ya que en este proyecto se pretende estudiar e implementar una red homogénea que integre una muestra de la red T con una de la red B.





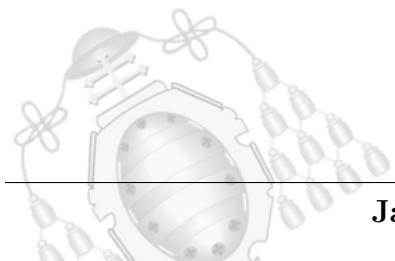
4. Trabajo realizado

En este capítulo se describe la parte práctica realizada en este proyecto. Para facilitar la comprensión del contenido se realiza una división de todo el proyecto en tres fases. La primera fase es la que engloba las etapa de investigación y familiarización con toda la tecnología de blockchain. Esta etapa se ve reflejada con el contenido de los capítulos anteriores. La segunda fase refleja el análisis de la Red B y la Red T de manera independiente, así como de los clientes Geth y Besu. Finalmente, la tercera fase consiste en buscar la interoperabilidad entre estos dos tipos de clientes. Las fases dos y tres son las que se documentan a lo largo de este capítulo.

4.1.- Entorno de desarrollo

Primero, se describe el entorno de desarrollo que se utiliza para desplegar una red de prueba de los tipos de Red B y Red T de Alastria. Estas redes se despliegan en dos máquinas virtuales de Ubuntu 18.04 LTS, utilizando como hipervisor el software VirtualBox. Además, como equipo anfitrión se utiliza un portátil con 8 núcleos, 16 Gb de memoria RAM y 512 Gb de disco de estado sólido para el almacenamiento. Es importante destacar que ambas máquinas virtuales tienen que tener asignada una capacidad de almacenamiento superior a los 30 Gb. Esto se debe a que cada una implementa varios nodos de la red de la blockchain, por tanto, como se ha explicado anteriormente, dichos nodos deben almacenar la información contenida en la cadena de bloques, y por eso es necesaria esa elevada capacidad de almacenamiento.

Por otro lado, también es esencial destacar que ambas máquinas virtuales deben tener comunicación entre ellas, ya que para poder sincronizarse. Además, para realizar la instalación de cada tipo de red en cada una de ellas también es indispensable que dichas máquinas virtuales tengan conexión a Internet.





4.2.- Instalación de la Red T

El primer paso que se realiza en el desarrollo de la parte práctica de este proyecto es la instalación de la Red T de prueba. Esta red tiene unas características similares a la Red T original de Alastria, pero en este caso está pensada para trabajar con ella en un entorno privado y poder realizar pruebas de desarrollo e implementación sobre la misma.

Recordando que es la Red T, según lo dicho en el apartado 3.3.2.2, es la red inicial de Alastria. Esta red está basada en la tecnología de Quorum, que se trata de una red de blockchain permisionada con características muy adecuadas para el uso empresarial.

Para llevar a cabo la instalación de esta red, se han seguido las indicaciones dadas por Alastria en [37]. En esta referencia se pueden encontrar tres formas diferentes para realizar la instalación de esta red. Dichos tres métodos son:

- Utilizar una testnet predefinida: Esta opción consiste en utilizar una testnet que permite ejecutar tres nodos directamente. Es fácil de implementar pero no aporta flexibilidad a la hora de poder añadir más nodos a la red. La implementación de esta red se realiza mediante la ejecución de unos scripts que permiten generar los tres nodos ya mencionados.
- Utilizar una red ya montada encapsulada en un contenedor Docker: La herramienta Docker permite crear y definir entornos de desarrollo portables. Al igual que el caso anterior, una vez se tiene controlado el entorno de Docker, es fácil desplegar esta red, pero sigue teniendo poca flexibilidad a la hora de definir los nodos.
- Utilizar una red creada desde cero: En este caso, se define la red completamente. Se realiza la implementación mediante la ejecución de una serie de scripts que permiten flexibilizar el número de nodos que se añaden a la red.

La tercera de las opciones es la que se usa, ya que es la que permite tener un mayor control sobre la red. Además, también es la opción que permite crear mayor cantidad de nodos. Para la instalación de esta red, se han seguido los pasos mostrados en la referencia [37]. Estos pasos, se resumen de manera abreviada a continuación:



1. Descargar el directorio de GitHub [37] desde la máquina virtual utilizada como Host para la Red T.
2. Una vez descargado, se debe ejecutar el script *bootstrap.sh*, que permite instalar todas las dependencias necesarias para poder ejecutar la red.
3. Tras instalar todas las dependencias, se ejecuta el script *start_network.sh*. A este script se le pasa como parámetro dos números. El primero identifica el número de nodos validadores que se quieren añadir a la red y el segundo el número de nodos regulares que se añaden a la red. Desde este punto, ya se consigue una red de prueba similar a la Red T utilizada por Alastria.
4. Mediante la ejecución del script *start_node.sh* se pueden ejecutar más nodos, ya sean nodos generales o reguladores, con el fin de aumentar el tamaño de la red. Pero en este caso, al ejecutar sobre un mismo ordenador todos los nodos, solo se ejecutan los comentados en el paso anterior.
5. Por último, se ejecuta el script *start_ethstats.sh*. Este permite realizar una monitorización de los principales parámetros de la red desde un navegador web.

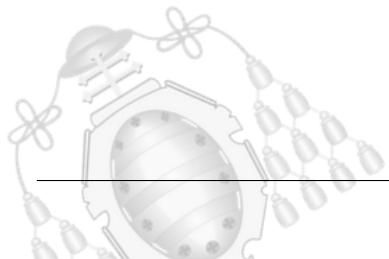
Además, se adjunta la figura 4.1 a modo de esquema del árbol de directorios que se utiliza para la ejecución de esta Red T de prueba. De este árbol de directorios, se destaca lo siguiente:

- Dentro de la carpeta *bin*, se encuentran los scripts ya mencionados anteriormente y que se utilizan para la ejecución de la red y del sistema de monitorización que implementa dicha red.
- En la carpeta *identities*, están los archivos que definen las identidades de los nodos que se pueden generar en la red. En este caso, las identidades de dichos nodos pueden ser general o validadores, y de manera predefinida hay cuatro identidades de cada tipo. Además, la red da la opción de que el usuario pueda añadir más identidades, es decir, más nodos de cada uno de los dos tipos.
- En la carpeta *network* se encuentra la información más relevante respecto a cada nodo. Dentro de esta carpeta, cada una de las identidades creadas tiene su propia carpeta en la que se almacenan sus claves privadas y claves públicas, así como



todo la información que almacena de la cadena de bloques. Además, en el caso de los nodos generales, en este directorio se encuentra la información relacionada con el servicio de *constellation*. Este servicio es utilizado para realizar transacciones privadas entre dos nodos o más nodos. Cuando se habla de transacciones privadas, se hace referencia a aquellas transacciones en las que solo los participantes de la misma conocen los datos de dicha transacción. De esta forma, se puede mantener totalmente la confidencialidad de la información intercambiada, ya que ningún nodo externo de la red puede ver esa información. Esto es una herramienta de gran utilidad desde el punto de vista empresarial, ya que permite intercambiar información muy sensible de forma segura y privada.

- Por último, es imprescindible destacar el fichero *permissioned.json*. Este fichero es utilizado para almacenar los enodes (concepto que se explica a continuación) de los nodos que tienen permiso para unirse a la red. Es decir, este fichero es la principal forma de implementar el permisionado de este tipo de redes. Todos los nodos, cuyo enode no esté registrado en este fichero, no se pueden conectar a la red.



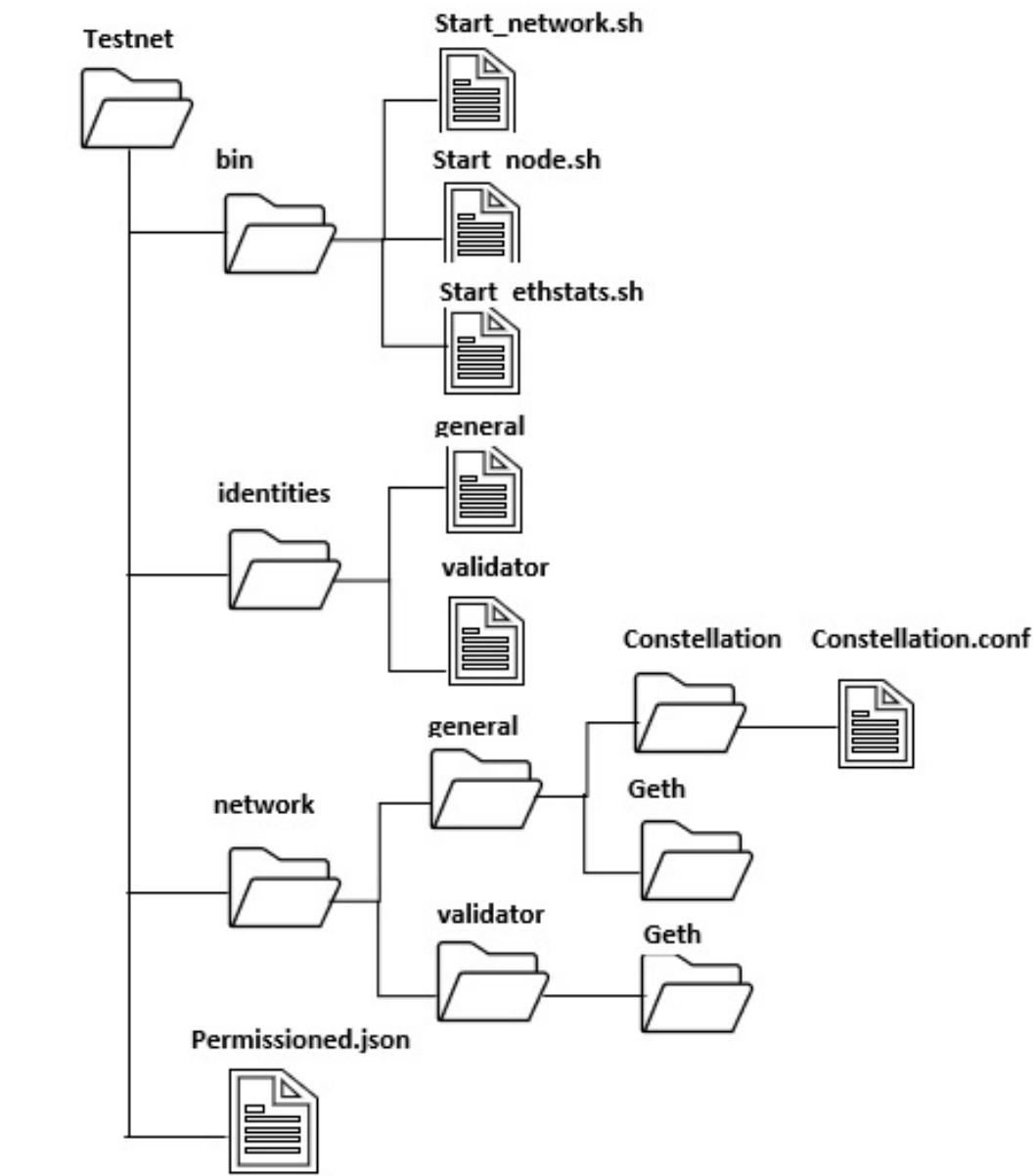
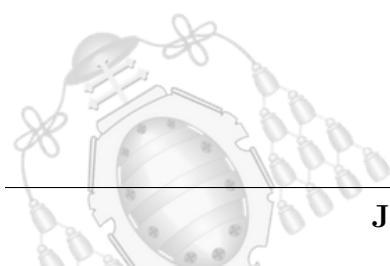


Figura 4.1.- Árbol de directorios utilizado para ejecutar la Red T

4.2.1.- Nodos de la red T

Para lograr entender por completo el funcionamiento de esta red y por tanto, el propósito de este proyecto, es imprescindible comprender cómo se implementan cada uno de estos nodos, cómo se identifican entre ellos y cómo se conectan.





En el mundo de la tecnología de blockchain se utiliza un tipo de software denominado clientes. Estos clientes permiten implementar los nodos de la red de blockchain y son la herramienta necesaria para interactuar con la red.

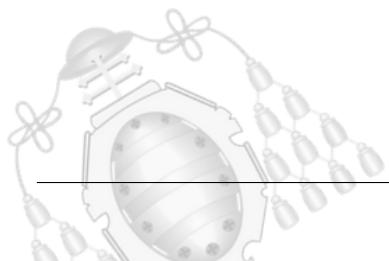
La Red T está basada en la tecnología de Quorum, que a su vez, tiene su origen en un fork de la cadena de bloques de la red Ethereum. Por dicho motivo, los clientes utilizados para interactuar con la red Ethereum, son también válidos para trabajar con la Red T de Alastria [38]. En la actualidad, existen una serie de clientes que son aptos para interactuar con la red Ethereum, estos son: Geth, Parity o Hyperledger Besu. En el caso particular de la Red T de Alastria, se utilizan clientes Geth, mientras que para la Red B se utiliza Hyperledger Besu. Las diferencias entre clientes es lo que conlleva a generar diferentes tipos de redes adaptadas a cada cliente. La Red T para clientes Geth y la Red B para clientes Besu. De esto surge el objetivo principal de este proyecto: buscar la interoperabilidad entre estas redes, para que funcionen de manera homogénea en una única red.

4.2.1.1.- Enode

Dentro del apartado de los nodos de la Red T, es necesario comentar qué es el enode. Aunque este concepto también es realmente útil para la Red B que se detalla en apartados posteriores.

El concepto de enode hace referencia al identificador que tiene cada nodo. Este identificador está formado por tres partes diferentes:

- ID: Este parámetro es una cadena de caracteres hexadecimales generados a partir de la clave privada que tiene ese nodo.
- Dirección IP: Hace referencia a la dirección IP en la que se está ejecutando el nodo.
- Puerto: Puerto UDP que utiliza el nodo para descubrir el resto de nodos de la red.





Generalmente, los enodes se representan basándose en el siguiente formato:

enode : //id@DireccionIP : Puerto

4.2.1.2.- Cliente Geth

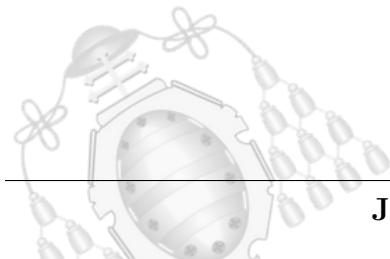
Es el cliente utilizado en la Red T original de Alastria, por tanto, la Red T de prueba instalada también utiliza clientes Geth para implementar los nodos y poder interactuar con ellos. Por dicho motivo, es importante entender sus características y su funcionamiento.

El nombre de Geth proviene de Go-Ethereum. Se trata del cliente más habitual en las redes de blockchain basadas en Ethereum. Este tipo de cliente se encarga de gestionar toda la interacción (creación de contratos inteligentes, minado de bloques, etc) mediante el uso de una línea de comandos [38].

Primero, para entender el funcionamiento de Geth, se debe analizar la línea de comandos que incorpora. Esta línea de comandos es utilizada en el momento de ejecución del cliente y permite añadir diversas opciones a la ejecución de este. De esta forma, los diferentes nodos que se ejecutan no tienen por qué tener todos la misma configuración. Algunas de las opciones que se pueden modificar mediante esta línea de comandos están relacionadas con: configuración de la red, APIs disponibles para conectarse, conexión con el resto de nodos, valor de Gas utilizado [39]. En la figura 4.2 se remarca un ejemplo de cómo utilizar la línea de comandos del cliente Geth para la ejecución del mismo. En la línea remarcada se ve cómo se ejecuta el cliente Geth y se utilizan algunas opciones como por ejemplo: *-datadir*. Esta opción permite elegir el directorio raíz para que se almacene la información de la cadena.

```
OTHER_NODES="`cat ${PWD}/identities/CONSTELLATION_NODES`"
GLOBAL_ARGS="--networkid $NETID --identity $IDENTITY --rpc --rpccaller 0.0.0.0 --rpcauth admin,db,eth,debug,miner,net,shh,txpool,personal,web3,q$"
CONSTELLATION_PORT="1800$PUERTO"
if [ "$NODE_NAME" == "main" -o "$NODE_NAME" == "validator1" -o "$NODE_NAME" == "validator2" ]; then
    nohup geth --datadir "${PWD}"/network/"$NODE_NAME" ${GLOBAL_ARGS} --mine --minerthreads 1 --syncmode "full" 2> "${PWD}"/logs/quorum_$$
else
```

Figura 4.2.- Ejecución de Geth mediante la línea de comandos





Por otro lado, también es indispensable comentar la consola de JavaScript que incorpora el cliente Geth. Esta consola permite realizar consultas y obtener información sobre la red a la que está conectado el nodo que es implementado mediante el cliente Geth. Además, también es importante destacar que esta consola de JavaScript puede ejecutarse de manera interactiva o de manera no-interactiva, es decir, mediante el uso de Script. Esta versatilidad a la hora de su uso, permite que sea una herramienta muy interesante en diversos ámbitos [40].

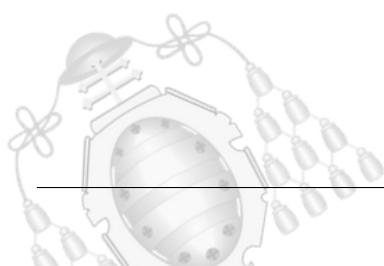
En la figura 4.3 se muestra un ejemplo de inicio de la consola interactiva de un cliente Geth. Y en la figura 4.4 se muestra un ejemplo de consulta mediante el cliente Geth. En este caso, se realiza una consulta para obtener información sobre el nodo. Se ha remarcado el comando utilizado, ya que es lo que permite obtener esta información. De todas formas, los parámetros presentes en dicha figura se explican en apartados posteriores.

```
ubuntu1@ubuntu1-VirtualBox:~$ geth console
WARN [06-14|10:56:12] No etherbase set and no accounts found as default
INFO [06-14|10:56:12] Starting peer-to-peer node
INFO [06-14|10:56:12] Allocated cache and file handles
INFO [06-14|10:56:12] Initialised chain configuration
  instance=Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4
  database=/home/ubuntu1/.ethereum/geth/chaindata
  config=[ChainID: 1 Homestead: 1150000 DAO: 1920000 DAOSupport: true EIP150: 24
  63000 EIP155: 2675000 EIP158: 2675000 Byzantium: 4370000 IsQuorum: false Engine: ethash]
INFO [06-14|10:56:12] Disk storage enabled for ethash caches
  dir=/home/ubuntu1/.ethereum/geth/ethash
  count=3
INFO [06-14|10:56:12] Disk storage enabled for ethash DAGS
  dir=/home/ubuntu1/.ethereum
  count=2
INFO [06-14|10:56:12] Initialising Ethereum protocol
INFO [06-14|10:56:12] Loaded most recent local header
  number=0 hash=d4e567..cb8fa3 td=17179869184
INFO [06-14|10:56:12] Loaded most recent local full block
  number=0 hash=d4e567..cb8fa3 td=17179869184
INFO [06-14|10:56:12] Loaded most recent local fast block
  number=0 hash=d4e567..cb8fa3 td=17179869184
INFO [06-14|10:56:12] Loaded local transaction journal
  transactions=0 dropped=0
INFO [06-14|10:56:12] Regenerated local transaction journal
  transactions=0 accounts=0
INFO [06-14|10:56:12] Starting P2P networking
INFO [06-14|10:56:12] UDP listener up
  self=enode://dfd96fb7f6de4b30fb9e5fe9d7e014014957f651942c9e46273eb463aa94c0d8d
  77665f04c673aabab71cb543812ff88106ed872df136eb31b8dbbb5538b3a@92.185.221.150:30303
INFO [06-14|10:56:14] RLPx listener up
  self=enode://dfd96fb7f6de4b30fb9e5fe9d7e014014957f651942c9e46273eb463aa94c0d8d
  77665f04c673aabab71cb543812ff88106ed872df136eb31b8dbbb5538b3a@92.185.221.150:30303
INFO [06-14|10:56:14] IPC endpoint opened: /home/ubuntu1/.ethereum/geth.ipc
INFO [06-14|10:56:14] Mapped network port
  proto=udp extport=30303 intport=30303 interface="UPNP IGDv2-IP2"
Welcome to the Geth JavaScript console!

instance: Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4
modules: admin:1.0 debug:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

> INFO [06-14|10:56:15] Mapped network port
  proto=tcp extport=30303 intport=30303 interface="UPNP IGDv2-IP2"
>
```

Figura 4.3.- Inicio de la consola interactiva de Geth





```
[admin.nodeInfo]
{
  enode: "enode://fdf960fb7f6de4b30fb9e5fe9d7e014014957f651942c9e46273eb463aa94c0d8d77665f04c673aabab71cb543812ff88106ed872df136eb31b8dbbb5538b3a",
  id: "fdf960fb7f6de4b30fb9e5fe9d7e014014957f651942c9e46273eb463aa94c0d8d77665f04c673aabab71cb543812ff88106ed872df136eb31b8dbbb5538b3a",
  ip: "92.185.221.150",
  listenAddr: "[::]:30303",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  ports: {
    discovery: 30303,
    listener: 30303
  },
  protocols: {
    eth: {
      difficulty: 17179869184,
      genesis: "0xd4e56740f876ae8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      head: "0xd4e56740f876ae8c010b86a40d5f56745a118d0906a34e69aec8c0db1cb8fa3",
      network: 1
    }
  }
}
```

Figura 4.4.- Consulta en la consola interactiva de Geth

Además, también se debe destacar que el cliente Geth permite el uso de JSON-RPC-APIs. Este conjunto de APIs permiten obtener información de la red y de los propios nodos, igual que la consola de JavaScript, pero esta vez a través de llamadas de procedimientos remoto (RPC) que se ejecutan sobre HTTP o WebSockets. Los métodos RPC utilizados para obtener información, se agrupan categorías en función de su propósito. Algunas de las categorías más importantes son: Admin, Eth, Personal, Miner [41].

4.2.2.- Configuración de la red

El objetivo principal del proyecto es estudiar la posible interoperabilidad entre las Red T y la Red B, de forma que al final puedan funcionar de manera homogénea. Para conseguir esto, a parte de implementar inicialmente una red de cada uno de estos tipos, se debe estudiar y analizar todos los parámetros de configuración y el funcionamiento más específico de cada una de las redes utilizadas. Esto permite obtener los conocimientos necesarios para que, cuando se intente realizar la integración de ambas redes, se conozcan los parámetros que se deben modificar. Por dicho motivo, primero se analizan los scripts utilizados.

4.2.2.1.- Script: start_network

La función principal de este script es recibir como parámetro el número de nodos validadores y nodos regulares que se quieren utilizar inicialmente en la red e invocar



al script *start_node* tantas veces como número de nodos se le pasen como parámetro. Además, en el caso de la inicialización de los nodos validadores, tiene una función adicional que se ve reflejada en la figura 4.5. Esto está ligado con el algoritmo de consenso utilizado en Quorum, es decir, el algoritmo de consenso IBFT. Si se recuerda lo mencionado en el capítulo 2, IBFT se basa en el funcionamiento de una máquina de estados, en la que las transiciones entre estados se realizan cuando se llega a un acuerdo de dos tercias partes de los nodos validadores. En este caso, se sigue manteniendo la idea de la aprobación por parte de dos tercias parte de los nodos, pero en este caso, lo que se propone es que los nuevos nodos validadores sean añadidos o eliminados del grupo de nodos validadores.

Se debe destacar que los comandos remarcados en la figura 4.5 pertenecen a un comando de la consola No-Interactiva del cliente Geth, dicho comando pertenece a la API *istanbul* disponible para el cliente Geth [42]. Esta API está relacionada con el funcionamiento del algoritmo de consenso IBFT (Istanbul Byzantine Fault Tolerant). Para utilizar este comando, se le pasan dos parámetros, una cadena y un booleano. El primer parámetro, representa la dirección del nodo, se trata una cadena de 40 valores hexadecimales mientras que el segundo se trata de un booleano que indica si ese nuevo nodo se quiere añadir (true) o se quiere borrar (false). Además, se puede observar que en cada ejecución de este comando se le asigna en cada iteración a uno de los nodos ya creados mediante la orden *attach* [43].

```
start_validators () {
    VAL_NUM=$1
    echo "[*] Starting validator nodes"
    if [ "$VAL_NUM" -eq "1" ]; then
        ./bin/start_node.sh main
    elif [ "$VAL_NUM" -eq "2" ]; then
        ./bin/start_node.sh main
        start_faulty_validator
        sleep 15
        geth --exec 'istanbul.propose("0xB50001FFA410F4D03663D69540c1C8e1C017e7e6", true)' attach network/main/geth.ipc
    elif [ "$VAL_NUM" -eq "3" ]; then
        ./bin/start_node.sh main
        start_faulty_validator
        sleep 15
        geth --exec 'istanbul.propose("0xB50001FFA410F4D03663D69540c1C8e1C017e7e6", true)' attach network/main/geth.ipc
        ./bin/start_node.sh validator2
        sleep 15
        geth --exec 'istanbul.propose("0xD8CfEA3B26B879f9D208975dFE8460F27520876b", true)' attach network/main/geth.ipc
        geth --exec 'istanbul.propose("0xD8CfEA3B26B879f9D208975dFE8460F27520876b", true)' attach network/validator1/geth.ipc
    else
        echo "[!!] Number of validators not supported. Please contact @arochaga or any Alastria member for support"
        exit
    fi
}
```

Figura 4.5.- Fragmento del script *start_network*





Debido a que se trata de un concepto complejo de explicar e interpretar, se añade un esquema que plantea una perspectiva más visual de lo que consiste este proceso.

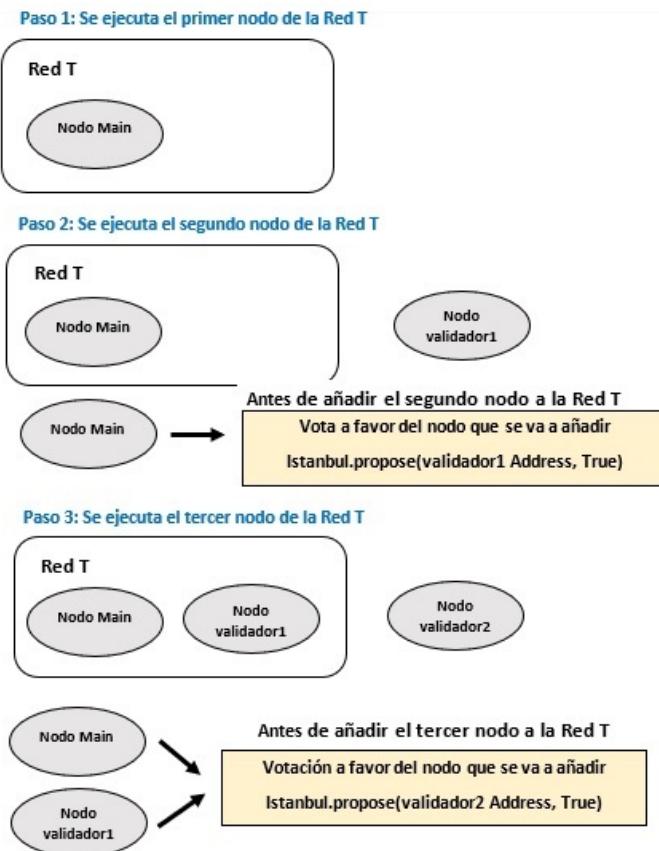
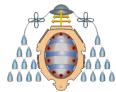


Figura 4.6.- Diagrama de funcionamiento de IBFT

Los pasos descritos en la figura 4.6, se pueden resumir de la siguiente forma:

1. Se ejecuta el nodo principal y se añade a la Red T
2. Se ejecuta el segundo nodo (validador1) y antes de añadirse a la red, se ejecuta la votación con los nodos ya existentes en la red, en este caso el nodo main. En este caso, el nodo main vota True y por tanto se añade ese nuevo nodo.
3. Se ejecuta el tercer nodo (validador2), antes de añadirse a la red, ese nodo se somete a la votación de los otros dos nodos (main y validador1). En este caso, está programado para que ambos voten a favor, entonces se añade ese nuevo nodo a la red.





4.2.2.2.- Script: start_node

Este script se invoca desde el script anterior tantas veces como nodos se pretenden añadir a la red. Es uno de los scripts más importantes dentro de la configuración de la Red T. En dicho script se definen la mayoría de parámetros de configuración, y que son esenciales para el funcionamiento de la red de blockchain. Para facilitar la comprensión de este script, se ha dividido en partes de forma que se expliquen de manera independiente para comprender mejor su importancia.

La primera de estas partes está relacionada con la declaración de las opciones de las líneas de comandos que se utilizan a la hora de ejecutar el cliente Geth. Esta parte se muestra en la figura 4.7.

```
OTHER_NODES=`cat ${PWD}/identities/CONSTELLATION_NODES`  
GLOBAL_ARGS="--networkId $NETID --identity $IDENTITY --rpc --rpcaddr 0.0.0.0 --rpcapi admin,cli,les,db,eth,debug,miner,net,shh,txpool,personal,web3,quorum,istanbul --rpport 2200$PUERTO"  
RPC_OPTIONS="--rpc --rpccaddr 0.0.0.0 --rpcaapi admin,cli,les,eth,debug,miner,net,txpool,personal,web3,quorum,istanbul --rpport 2200$PUERTO"  
GENERAL="--networkId $NETID --identity $IDENTITY --ethstats $IDENTITY:bb98a0b6442386d0cdf8a31b267892c1@$ETH_STATS_IP:3000"  
NETWORK="--port 2100$PUERTO"  
MINING="--mine --minerthreads 1 --syncmode \"full\""  
DATA_DIR="--datadir \"${PWD}/network/$NODE_NAME\""  
EXTRA="--miner.gaslimit 0000000"  
CONSTELLATION_PORT="900$PUERTO"
```

Figura 4.7.- Script start_node: Declaración de variables

Debido a que se necesitan utilizar muchas opciones a la hora de definir la configuración del cliente Geth, se decide organizarlas en variables, de forma que sea más sencillo y más eficiente trabajar con ello. A continuación se detallan las opciones utilizadas más importantes:

- **RPC_OPTIONS:** Está ligado con las opciones relacionadas con las comunicaciones mediante la API de JSON-RPC-HTTP. Las opciones utilizadas en este caso:
 - **Rpc:** Se activan las comunicaciones RPC.
 - **Rpccaddr:** Dirección IP en la que se reciben las comunicaciones RPC. En este caso, al utilizar la dirección 0.0.0.0 se permiten todas.
 - **Rpcaapi:** Sirve para definir las APIs que se permiten en ese nodo. Las más utilizadas son: ETH, NET o ADMIN.
 - **Rpport:** Indica el puerto abierto para las comunicaciones RPC. En este caso, al estar pensado para implementar varios nodos desde la misma





máquina, cada nodo creado tiene un puerto diferente. Los puertos utilizados en este caso son el rango: 2200X. Siendo X el valor que cambia en función del tipo de nodo.

- **GENERAL:** Incluye alguna de las opciones generales necesarias para realizar la configuración de la red.
 - **NetworkId:** Permite definir el identificador de red. Todos los nodos deben tener el mismo identificador de red, de lo contrario, no se pueden conectar a la misma red.
 - **Ethstats:** Sirve para definir la url utilizada para consultar las estadísticas de la red y que son visibles desde un navegador web. Es un parámetro que no influye desde el punto de vista de funcionamiento de la red, pero es imprescindible para la monitorización de esta.
- **DATA_DIR:** Sirve para definir el directorio donde se almacena la información de la red, y así tener una correcta organización de la misma.

Además, a parte de las opciones ya mencionadas, en el script se definen algunas otras que están relacionadas con el proceso de generación de bloques. No se profundiza en la explicación de estas porque no se consideran parte fundamental del proyecto.

Por otro lado, otra de las variables mostradas en la figura 4.7. Esta variable es CONSTELLATION_PORT. Está relacionada con el servicio de constellation que está presente dentro de la Red T. Este servicio, como ya se explicó anteriormente, está ligado a las transacciones privadas que se puede realizar en la red. Esta variable se utiliza para definir el puerto utilizado para cada nodo dentro de esta constellation. De todas formas, no se entra en más detalle en este concepto, al no ser un elemento fundamental en el desarrollo del proyecto.

La segunda parte del script start_node.sh está relacionada con la invocación del cliente Geth cuando se trata de un nodo validador o nodo main. Simplemente, la función de esta parte del script es comprobar que la identidad del nodo sea validador o main y en caso afirmativo ejecutar el comando Geth con las opciones descritas anteriormente. Además, también se define el fichero de logs correspondiente donde se almacena todos





los avisos que la ejecución del cliente Geth genera. Este segundo fragmento de script se muestra en la figura 4.8.

```
if [ "$NODE_NAME" == "main" -o "$NODE_NAME" == "validator1" -o "$NODE_NAME" == "validator2" ]; then
    nohup geth $DATA_DIR $RPC_OPTIONS $GENERAL $NETWORK $MINING 2 > "${PWD}"/logs/quorum_"$NODE_NAME"_"${_TIME}".log &
```

Figura 4.8.- Script start_node: Invocación nodo validador

Finalmente, la tercera parte de este script es similar a la segunda parte, pero en lugar de tratar la invocación del cliente Geth cuando se pretende crear un nodo validador se trata la invocación del cliente Geth cuando se pretende crear un nodo regular. Además, incorpora también la creación del fichero constellation.conf. Este fichero es utilizado por los nodos regulares para definir los parámetros fundamentales que influyen en el servicio de constellation. Esta tercera parte del script start_node.sh se muestra en la figura 4.9.

```
else
    # TODO: Add every regular node for the constellation communication
    generate_conf "${NODE_IP}" "${CONSTELLATION_PORT}" "${OTHER_NODES}" "${PWD}"/network "$NODE_NAME" > "${PWD}"/network/"$NODE_NAME"/constellation.conf
    PWD="${pwd}"
    nohup constellation-node "${PWD}"/network/"$NODE_NAME"/constellation/conf 2>> "${PWD}"/logs/constellation_"$NODE_NAME"_"${_TIME}".log &
    check_port ${CONSTELLATION_PORT}
    nohup env PRIVATE_CONFIG="${PWD}"/network/"$NODE_NAME"/constellation/conf geth --ipcpath $(pwd) --datadir "${PWD}"/network/"$NODE_NAME"_"${_TIME}".log &
fi
```

Figura 4.9.- Script start_node: Invocación nodo regular

4.2.3.- Resumen funcionamiento Red T

En este apartado se adjuntan una serie de imágenes que permiten corroborar el correcto funcionamiento de la red, así como comprobar varias de las características de las redes de blockchain comentadas a lo largo de este proyecto.

En la primera de estas imágenes, figura 4.10, se observan las estadísticas de una red formada por tres nodos. En este caso se han implementado dos nodos validadores (validator1 y validator2), además del nodo main, que no deja de ser otro nodo validador más. De esta primera imagen se puede destacar que en cada nodo se visualiza que tiene 2 nodos conectados a él. De esta forma se verifica que la red está funcionando correctamente y que existe conexión entre todos los nodos.



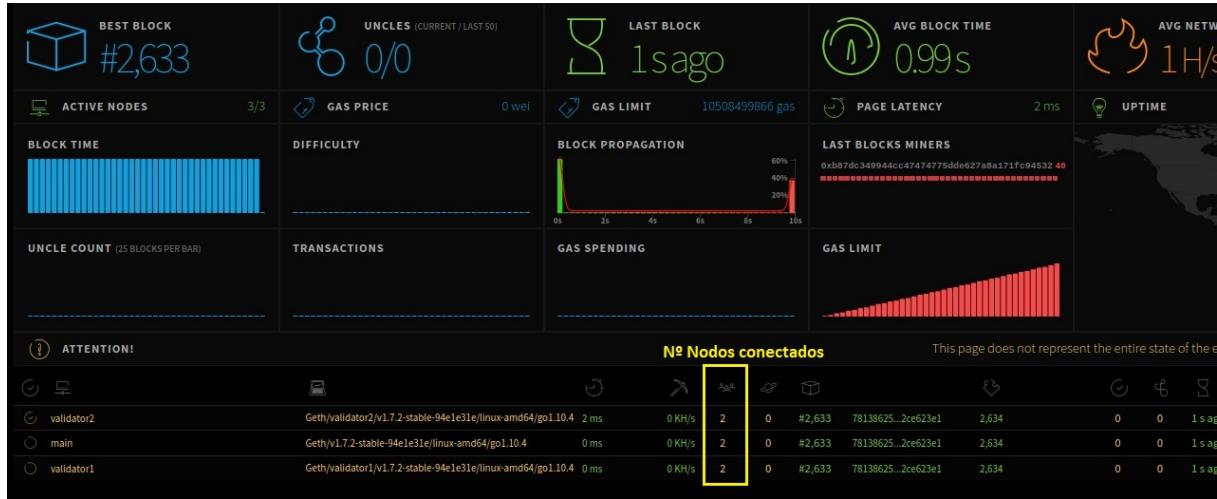
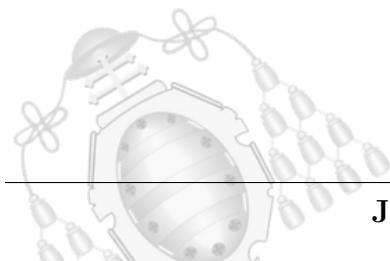


Figura 4.10.- Ejemplo de una red con tres nodos

En la siguiente imagen, figura 4.11, se muestra la consola interactiva de JavaScript del nodo *validator1*. En este caso, se ha utilizado la API *admin*. Esta API ha sido habilitada a la hora de la ejecución del nodo. En este caso, el comando utilizado es *Admin.peers*, el cual permite conocer la información de los nodos que están conectados al nodo desde el que se ejecuta, en este caso el nodo *validator1*. A continuación se enumeran y explican los campos mostrados en esta figura:

- Se muestra la información relativa a los dos nodos que están conectados al nodo *validator1*. Se observa que el nombre de esos dos nodos son: *Main* y *validator2*. Es decir, esto se corresponde con la configuración planteada inicialmente de esta red.
- Dentro de la información relativa a cada nodo (*validator2* y *main*) se puede destacar lo siguiente:
 - Utilizan el algoritmo de consenso Istabul. Esto se refiere al algoritmo de consenso IBFT (Istanbul Byzantine Fault Tolerant).
 - El campo *id* representa parte del enode de cada uno de estos nodos.
 - También se refleja la información relativa a los puertos utilizados para realizar la conexión con cada uno de los nodos.





```
[admin.peers] Comando utilizado
[{
  caps: ["istanbul/64"],
  id: "9ddee7e76c5ea204b25c6e7473f3617ec5d01571b772ed0e271c3adc6d78ef930decdb27ea469db7be46dc449c8f24d2bcd5e48105be7a58f953f9eaaf24582e",
  name: "Geth/validator2 v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    Nombre de los nodos conectados
    localAddress: "[::1]:32794",
    remoteAddress: "[::1]:21006"
  },
  protocols: {
    istanbul: {
      difficulty: 1030,
      head: "0x76b6a37f93398befd2969775cea9746f101fc387f4b8c209473b8166cb0009ce",
      version: 64
    }
  }
}, {
  caps: ["istanbul/64"],
  id: "3905f943ba5446eba164c07ab5f53a84ce17d74ec4d7591f6ec54b9d7608f57cae7cfdf946616385f59cfb5b910161a1f8520cb6f992bcc0d1ab932601205e91",
  name: "Geth/main v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    Nombre de los nodos conectados
    localAddress: "[::1]:51800",
    remoteAddress: "[::1]:21000"
  },
  protocols: {
    istanbul: {
      difficulty: 1027,
      head: "0x80ee5c72792278fc184e1cc42b321f6ee9086e684fce066acead7760565bae14",
      version: 64
    }
  }
}]
```

Figura 4.11.- Información de los nodos de la red (1)

En la figura 4.12, se muestra la misma información que en la figura 4.11, pero esta vez desde el punto de vista del nodo validator2.

```
[admin.peers] Comando utilizado
[{
  caps: ["istanbul/64"],
  id: "b2da00dfe8431813a86f4f08637ea2edf3108d3505cd03bef30c791ac9236e02fa0cce569c275f7c1a417ff088f68cb619e97586154254424afe2f1e60843a",
  name: "Geth/validator1 v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    Nombre de los nodos conectados
    localAddress: "[::1]:21006",
    remoteAddress: "[::1]:32794"
  },
  protocols: {
    istanbul: {
      difficulty: 1566,
      head: "0x83409804114cb493b6d34deff42ce1acf1774d518de48734adcf1c1bedfee1f",
      version: 64
    }
  }
}, {
  caps: ["istanbul/64"],
  id: "3905f943ba5446eba164c07ab5f53a84ce17d74ec4d7591f6ec54b9d7608f57cae7cfdf946616385f59cfb5b910161a1f8520cb6f992bcc0d1ab932601205e91",
  name: "Geth/main v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    Nombre de los nodos conectados
    localAddress: "[::1]:51812",
    remoteAddress: "[::1]:21000"
  },
  protocols: {
    istanbul: {
      difficulty: 1571,
      head: "0x64ce4ff11abe6d3d125fea9004e9d2c299133b8c9ab2da73aedd738427ef0fc1",
      version: 64
    }
  }
}]
```

Figura 4.12.- Información de los nodos de la red (2)

Por otro lado, también es interesante comprobar como en las figuras 4.11 y 4.12, el nodo Main tiene el mismo valor del campo id. Esto demuestra que la red está funcionando correctamente y que todos los nodos tienen conexión con los otros.

A continuación, también se analiza la forma en la que se generan bloques de la cadena, ya que esto también representa una parte esencial del funcionamiento de la



tecnología de blockchain. Para verificar el correcto funcionamiento, se adjuntan tres imágenes que representan la información de tres bloques consecutivos en la cadena. De esta forma, se puede comprobar la relación entre los Hashes de los bloques. La información de estos bloques es obtenido a partir de la API Eth. Esta API es otra de las habilitadas a la hora de la ejecución del nodo Geth.

En cada una de las tres imágenes, hay tres filas remarcadas. Esta información es la referida a:

- El Hash del bloque actual
 - El número de bloque
 - El Hash del bloque anterior

A partir de estos datos, se puede comprobar lo explicado en el capítulo 2. El Hash que tiene cada bloque, sirve como identificador del mismo dentro de la cadena, además de ser el nexo de unión con el bloque anterior y el bloque posterior.

En la figura 4.13 se muestra la información del primero de los tres bloques mencionados. En este caso se trata del bloque número 1743. Este bloque se ha elegido de manera aleatoria, una vez que se ha puesto en marcha la Red de Blockchain.

Figura 4.13.- Información sobre el bloque 1743

En la figura 4.14 se muestra la información del siguiente bloque. Se trata del bloque número 1744. En dicha figura se pueden ver resaltados los tres datos mencionados

anteriormente. De forma que se puede comprobar fácilmente como el Hash del bloque 1743 se encuentra en el campo *ParentHash* del bloque 1744.

Figura 4.14.- Información sobre el bloque 1744

La figura 4.15 representa la información del bloque 1745. En esta figura se pueden observar los datos mencionados anteriormente y la relación de Hashes con los bloques anteriores.

Figura 4.15.- Información sobre el bloque 1745

4.3.- Instalación de la Red B

Como segundo paso del desarrollo de la parte práctica de este proyecto, se encuentra el despliegue de una red de prueba similar a la Red B de Alastria. Junto al despliegue



de la Red T, sirve como base para el objetivo final del proyecto, es decir, para conseguir la interoperabilidad entre la Red B y la Red T.

Para realizar la instalación de esta red de prueba, se han contemplado las diferentes opciones que se ofrecen en la página oficial de Hyperledger Besu. Se ha escogido como opción final la que se corresponda con la referencia [44]. En esta referencia se puede encontrar información asociada a la implementación de una red privada similar a la Red B de Alastria, usando como algoritmo de consenso IBFT 2.0. Se ha elegido esta opción ya que es la más cercana a lo que se necesita en el futuro a la hora de buscar la interoperabilidad de ambas redes.

4.3.1.- Procedimiento de instalación Red B

Para realizar la instalación de esta red, se sigue el proceso mostrado en la referencia [44]. El proceso de instalación se resume a continuación:

1. Como prerequisito a la instalación de la Red, se necesita instalar Besu en la máquina sobre la que se realiza la instalación de la red. Para descargar Besu se utiliza la referencia [45].
2. Posteriormente, se realiza la creación del árbol de directorios donde se almacena la información relativa a cada nodo. Cabe destacar que para que el algoritmo de consenso elegido (IBFT 2.0) funcione correctamente, se necesitan al menos cuatro nodos validadores.
3. Se definen los ficheros de configuración y el archivo génesis de la red. Esto permite tener la configuración fundamental de la red y es la que usan todos los nodos implementados en esta.
4. Se ejecuta el primer nodo de la red, utilizando las opciones de la línea de comandos de Besu. De este primer nodo, es interesante tomar nota del enode que presenta, ya que este enode es fundamental para la ejecución de los siguientes nodos.
5. Se ejecutan el resto de nodos, utilizando el enode del primer nodo ejecutado como referencia, para que todos los demás nodos se conecten a él y se puedan sincronizar.



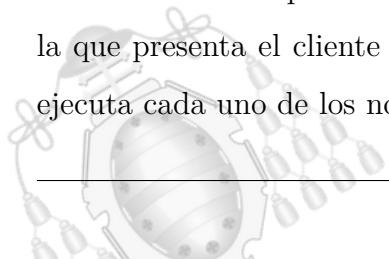
A partir de estos pasos, ya se tiene la Red B de prueba en funcionamiento. Más adelante, se detalla con mayor exactitud todo el procedimiento utilizado para la instalación de la Red B de prueba.

4.3.2.- Cliente Hyperledger Besu

En este apartado se detalla el funcionamiento y las principales características del cliente utilizado para la implementación de la Red B. Se trata del cliente Hyperledger Besu. Es un cliente de código libre, desarrollado por la fundación Hyperledger y que está desarrollado en Java [46]. Las principales fortalezas de este cliente son [47]:

- **Permisionado seguro:** Presentan una capacidad de aplicar el permisionado de manera mucho más flexible que en el caso de la Red T. Con este cliente, el permisionado se puede aplicar solamente a unos nodos en concreto, o dentro de la propia cadena de bloques se le puede aplicar a una determinadas aplicaciones descentralizadas.
- **Algoritmo de consenso más eficiente:** Se utilizan algoritmos de consenso más innovadores. Un ejemplo es IBFT 2.0. Este algoritmo de consenso es una mejora del IBFT 1.0 utilizado por el cliente Geth.
- **Interoperabilidad con aplicaciones de desarrollo:** Cada vez que se trabaja con un cliente de blockchain, es fundamental que sea integrable con herramientas de desarrollo como pueden ser Remix o Truffle, que son las herramientas que generalmente se utilizan para programar y compilar los smart contracts.
- **Alto grado de privacidad:** Permite flexibilizar el uso de transacciones privadas. Para ello utiliza un nodo denominado ORION. Estas transacciones privadas permiten que la información enviada entre los participantes de la transacción no sea visible para el resto de participantes de la red. Desde el punto de vista empresarial es interesante, ya que permite el intercambio de datos que requieren un alto grado de confidencialidad.

También es importante destacar que cuenta con una línea de comandos similar a la que presenta el cliente Geth, ya que permite modificar las opciones con las que se ejecuta cada uno de los nodos [48]. Pero esta línea de comandos no es la única opción





para modificar la configuración de los nodos, ya que esto también se puede realizar mediante el uso de un fichero de configuración. Un ejemplo de la línea de comandos del cliente Besu se observa en la figura 4.16.

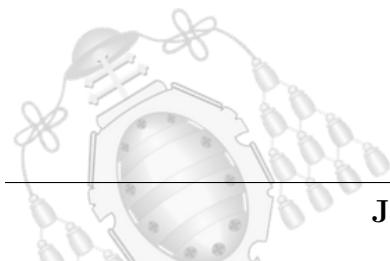
```
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/IBFT-network/Node4S ../../Besu/bin/besu --data-path=data --genesis-file=../genesis.json --bootnodes=enode://e2f8913d833c0cd4ba3906f3961e84e95aa6bf01de1318dee4f321df0fd4004904cf5c57ee5b38567cce091251293e68c8fc15e8c5b56b1e3319b8f051cff7f0@127.0.0.1:30303 --p2p-port=30306 --rpc-http-enabled --rpc-http-apl=ETH,NET,IBFT --host-whitelist="*" --rpc-http-cors-origins="all" --rpc-http-port=8548
2020-06-17 08:48:05.263+02:00 | main | INFO | Besu | Starting Besu version: besu/v1.4.1/linux-x86_64/openjdk-java-11
```

Figura 4.16.- Línea de comandos del cliente Besu

Además, al igual que el cliente Geth también presenta una API de JSON-RPC sobre HTTP que permite realizar peticiones, de forma que se pueda obtener información de la red o del propio nodo a partir de esta API [49]. En la figura 4.17 se muestra un ejemplo de uso de esta API para los cuatro nodos que forman la red. En este caso se utiliza el método *net_enode* que permite obtener el valor del enode del nodo sobre el que se realiza la consulta. Para determinar el nodo al que se realiza la consulta hay que fijarse en el puerto usado en la URL incluida al final del comando. Para el primer nodo se utiliza el puerto 8545 (puerto definido para las conexiones RPC para el primer nodo), para el segundo se utiliza el 8546, para el tercero el 8547 y para el cuarto 8548. Entonces, variando ese puerto, se puede elegir a qué nodo se le realiza la petición HTTP. Como se ve en la figura, el resultado obtenido para cada nodo es diferente, ya que el enode es un identificador único para cada uno.

```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8545
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "enode://e2f8913d833c0cd4ba3906f3961e84e95aa6bf01de1318dee4f321df0fd4004904cf5c57ee5b38567cce091251293e68c8fc15e8c5b56b1e3319b8f051cff7f0@127.0.0.1:30303"
}ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8546
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "enode://f45979cdcd2355a424df000b28e6df4c402610977298960fe624afbcea768f976fb2676792e53f09677a378116e6070edc029a95b77cb4fdd040f86ea182559@127.0.0.1:30304"
}ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8547
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "enode://c90cf19a5d290960a1be5a29515b0a077a5a5dd96dffcf5705dfef48451df99aa9f1981cd95355788f32ad2e5c097269bcb0a1ca8ab1c64d31d2db3979e2b383@127.0.0.1:30305"
}ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8548
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : "enode://bb65e98425e5dc07641f3ee1a8f1ca8a161c9077a890155e68debb33429a891ea4a5de27d5e75a182a5ff48e0ecbb47151a71e66ee051fbdd6fec6bd0c02f5cc@127.0.0.1:30306"
```

Figura 4.17.- Línea de comandos del cliente Besu





4.3.3.- Configuración de la Red B

En esta sección del documento se detalla en mayor profundidad la configuración inicial realizada en la Red B y que ha servido para realizar el despliegue de una Red de blockchain privada utilizando la tecnología de Hyperledger Besu.

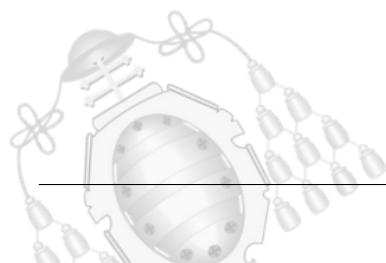
Esta red se ha montado eligiendo la opción de utilizar como algoritmo de consenso IBFT 2.0. Recordando en que consiste este algoritmo de consenso, se trata de un algoritmo en el que los nodos validadores votan para aceptar o no si se añade un nuevo nodo a la red. En caso de que dos terceras partes de los nodos validadores acepten la propuesta, este nuevo bloque se acepta. Es un algoritmo de consenso que no presenta un gran coste computacional.

Para la configuración inicial también se ha elegido la opción de utilizar un archivo génesis para definir el bloque cero de la cadena. Este tipo de archivos permiten definir las características básicas de la cadena. Además, es importante que todos los nodos que pretendan formar parte de una misma cadena utilicen el mismo archivo génesis, ya que de lo contrario, su configuración es diferente y por tanto se producen fallos a la hora de la sincronización. A continuación se muestra una imagen del fichero *genesis.json* utilizado en este caso.

```
GNU nano 2.9.3                                     genesis.json

{
  "config": {
    "chainId": 160616,
    "constantinoplefixblock": 0,
    "lbtft2": {
      "blockperiodseconds": 2,
      "epochLength": 30000,
      "requesttimeoutseconds": 10
    }
  },
  "nonce": "0x0",
  "timestamp": "0x58ee40ba",
  "gasLimit": "0x47b760",
  "difficulty": "0x1",
  "mixHash": "0x63746963616c2062797a616e74696e6520661756c7420746f6c6572616e6365",
  "coinbase": "0x0000000000000000000000000000000000000000000000000000000000000000",
  "alloc": {
    "fe3b557e8fb62b89f4916b721be55ceb828dbd73": {
      "privateKey": "8f2a55949038a9610f50fb23b5883af3b4ecb3c3bb792cbcefd1542c692be63",
      "comment": "private key and this comment are ignored. In a real chain, the private key should NOT be stored",
      "balance": "0xad78ebc5ac6200000"
    },
    "627306090abaB3A6e1400e9345bc60c78a8BEf57": {
      "privateKey": "c87509a1c067bde78beb793e6fa76530b6382a4c0241e5e4a9ec0a0f44dc0d3",
      "comment": "private key and this comment are ignored. In a real chain, the private key should NOT be stored",
      "balance": "90000000000000000000000000000000"
    }
  }
}
```

Figura 4.18.- Ejemplo de un archivo *genesis.json*





De los archivos génesis se puede destacar que al menos hay cuatro campos que es obligatorio que estén especificados en el fichero. Estos campos son: config, difficulty, gasLimit, alloc. Pero además de esos, hay varios campos más que se pueden utilizar. A continuación se detalla el propósito que tienen alguno de los campos más importantes [50]:

- **Config:** Se refiere a la configuración de la cadena de bloques. Dentro de este campo se encuentran varias opciones, algunas se explican a continuación:
 - **ChainID:** Representa el identificador de la cadena de bloques. En caso de tratarse de una red pública, este valor sería 1, pero en este caso, al tratarse de una red de prueba, se puede utilizar cualquier número entero positivo. En este caso se utiliza el 160616.
 - **ibft2:** Indica que se utiliza el algoritmo de consenso IBFT 2.0.
- **Nonce:** Es una cadena de 64 dígitos hexadecimales que provienen de una función Hash. Generalmente este campo es importante cuando se utilizan algoritmos de PoW. En este caso, al utilizarse el algoritmo de consenso IBFT 2.0, este campo pierde su utilidad y por eso tiene como valor 0 [51].
- **Timestamp:** Es una marca de tiempo que facilita la sincronización entre el nuevo bloque y los bloques añadidos previamente a la cadena, ya que si no siguen el mismo ciclo temporal, se incrementa la dificultad en la cadena de bloques.
- **GasLimit:** Representa el límite máximo de Gas que se puede gastar en completar un bloque. El total de Gas consumido en un bloque está relacionado con redes que llevan un coste asociado en Gas a la hora de realizar transacciones.
- **Difficulty:** Representa la dificultad que tiene la red para completar un nuevo bloque. Para redes de prueba se suele utilizar un valor bajo, de forma que los bloques se generen rápidamente y sin necesidad de utilizar muchos recursos.
- **MixHash:** Es un valor de 256 bits que se utiliza en combinación con el campo Nonce y que permite definir la cantidad de computación que se necesita.
- **Alloc:** Este campo se utiliza para predefinir cuentas en los nodos. De forma que si luego se quiere probar a realizar transacciones, se pueden utilizar esas cuentas predefinidas para realizar esas pruebas.





En esta red se ejecutan cuatro nodos validadores, de forma que primero es necesario crear un directorio para cada uno de ellos. En dicho directorio se almacena el contenido que los nodos descargan de la cadena de bloques.

Por otro lado, para la ejecución de dichos nodos se utilizan unos scripts programados en bash. De esta forma se pueda automatizar la ejecución de la red al máximo. El elemento principal de estos scripts es la ejecución del cliente Besu utilizando las diferentes opciones de su línea de comandos. Es importante destacar como el Nodo1 va a tener opciones de configuración diferentes a los Nodos 2, 3 y 4. Esto se debe a que el Nodo1 actúa como nodo Bootnode y el resto de nodos se conectan a él. En los próximos párrafos se detalla los scripts utilizados para la ejecución de estos nodos.

Para la puesta en funcionamiento del Nodo1 se utiliza el script mostrado en la figura 4.19. En este script, las diferentes opciones de la línea de comandos que se quieren utilizar se almacenan en variables y, posteriormente, se ejecuta el cliente Besu con dichas opciones.

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-

# Definición de variables con las opciones del cliente
OPTIONS_P2P="--p2p-enabled=true --p2p-host=192.168.1.35 --p2p-interface=0.0.0.0 --p2p-port=30303"
OPTIONS_RPC_HTTP="--rpc-http-enabled --rpc-http-host=0.0.0.0 --rpc-http-port=8545 --rpc-http-api=ETH,NET,IBFT,ADMIN --rpc-http-cors-origins=$"

# Ejecuto Nodo 1
cd Node1
gnome-terminal -e "cd ..;/Besu/bin/besu --data-path=data --genesis-file=../genesis.json --host-whitelist=\"*\" $OPTIONS_P2P $OPTIONS_RPC_HTTP"
```

Figura 4.19.- Script para ejecutar el Nodo 1 de la Red B

A continuación se explican brevemente las opciones de la línea de comandos del cliente Besu utilizadas en este caso:

- **Opciones P2P:** Son las opciones relacionadas con la comunicación P2P entre los nodos que forman la red. Las opciones utilizadas en este caso son:
 - **P2P-Enabled:** Se activa la comunicación P2P. En caso de estar desactivado, no se podrían descubrir los nodos vecinos.
 - **P2P-Host:** Dirección donde está funcionando el nodo. En este caso se utiliza la dirección IP privada de la máquina virtual. Al estar ejecutando los cuatro

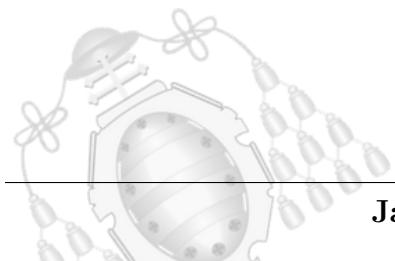




nodos sobre la misma máquina virtual, es válido también utilizar la dirección de loopback, 127.0.0.1.

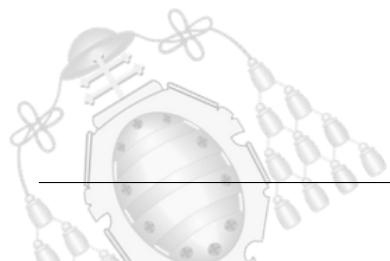
- **P2P-Interface:** Indica la interfaz en la que escucha el nodo. En este caso como no se requieren medidas de seguridad estrictas, se ha utilizado el valor 0.0.0.0 que indica que el nodo está escuchando en todas las interfaces.
- **P2P-Port:** Sirve para seleccionar el puerto que se quiere utilizar para realizar las comunicaciones P2P. Para este primer nodo, se utiliza el puerto 30303.
- **Opciones JSON-RPC:** Este segundo grupo de opciones están relacionadas con las APIs ejecutadas mediante RPC y que permiten obtener información sobre los nodos o la propia red mediante peticiones HTTP.
 - **Rpc-http-enabled:** Permite activar las comunicaciones mediante JSON-RPC-HTTP.
 - **Rpc-http-host:** Indica la dirección IP donde permite las conexiones. En este caso se utiliza la dirección 0.0.0.0 que es la que permite todas las conexiones.
 - **Rpc-http-Port:** Define el puerto utilizado por ese nodo para atender las peticiones de JSON-RPC-HTTP. Para este primer nodo se utiliza el puerto 8545.
 - **Rpcapi:** Define las API permitidas para utilizar con este nodo mediante JSON-RPC-HTTP. En este caso se han elegido: ETH, NET, ADMIN e IBFT.
- **Opciones generales:** En este tercer grupo se incluyen opciones más generales del funcionamiento de los nodos y que tienen diferentes propósitos.
 - **Data-dir:** Indica el directorio donde se almacenan los datos del nodo.
 - **Genesis-file:** Indica que archivo genesis.json se utiliza para definir el bloque 0 de la cadena de bloques.

En la figura 4.20 se muestra el script utilizado para la ejecución de los Nodos 2, 3 y 4 de la Red B de prueba. El funcionamiento de este script se detalla a continuación:





- Inicialmente, se definen las variables que almacenan diferentes opciones de la línea de comandos. Esto se hace de igual forma que para el script explicado anteriormente. Estas opciones de la línea de comandos están relacionadas con las comunicaciones P2P entre nodos y con la comunicación con la API JSON-RPC-HTTP.
- Posteriormente, se ejecuta una petición HTTP, utilizando la API JSON-RPC, al Nodo1. Dicha petición permite obtener el valor de su enode, ya que este valor es necesario para poder ejecutar los Nodos 2, 3 y 4. Para realizar esta petición se utiliza el método *Net_Enode* de la API NET. Además, de esta petición se puede destacar la URL utilizada para realizarla: <http://127.0.0.1:8545>, donde se utiliza el puerto 8545 para realizar la petición al Nodo1. Estos datos son acordes al parámetro definido en el script mostrado en la figura 4.19. Una vez realizada la petición HTTP, se obtiene el resultado en lenguaje JSON. Esta información se almacena en un fichero, después se manipula esa información para obtener únicamente el valor del enode correspondiente. Ese valor del enode, se almacena en una variable dentro del script, y posteriormente es utilizada como parámetro para ejecutar los clientes Besu.
- Finalmente, se ejecutan los tres nodos utilizando las opciones definidas previamente. De estos nodos se debe destacar que a pesar de que tienen muchas opciones de configuración comunes, las opciones de rpc-http-port y p2p-port son diferentes para cada uno de ellos, ya que al estar ejecutándose todos sobre la misma máquina, es imprescindible que tengan diferentes puertos, ya que de lo contrario se solaparían unos con otros.





```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-
Definición de las variables con las opciones comunes a los clientes

OPTIONS_P2P="--p2p-enabled=true --p2p-host=192.168.1.35 --p2p-interface=0.0.0.0"
OPTIONS_RPC_HTTP="--rpc-http-enabled --rpc-http-host=0.0.0.0 --rpc-http-api=ETH,NET,IBFT,ADMIN --rpc-http-cors-origins='all'"
GENERAL="--data-path=data --genesis-file=../genesis.json --host-whitelist='*'"

touch Info
Obtención del enode del Nodo1 ya ejecutado

echo $!curl -X POST --data '{"jsonrpc":"2.0","method":"net_enode","params":[],"id":1}' http://127.0.0.1:8545 > Info
ENODE=$(grep result Info | cut -d " " -f 3 | cut -d " " -f 4)

#Ejecuto Nodo 2
cd Node2
Ejecución del cliente Besu para el Nodo2
gnome-terminal -e ".../Besu/bin/besu $GENERAL --bootnodes=$ENODE --p2p-port=30304 --rpc-http-port=8546 $OPTIONS_P2P $OPTIONS_RPC_HTTP"

#Ejecuto Nodo 3
cd ..
Ejecución del cliente Besu para el Nodo3
cd Node3
gnome-terminal -e ".../Besu/bin/besu $GENERAL --bootnodes=$ENODE --p2p-port=30305 --rpc-http-port=8547 $OPTIONS_P2P $OPTIONS_RPC_HTTP"

#Ejecuto Nodo 4
cd ..
Ejecución del cliente Besu para el Nodo4
cd Node4
gnome-terminal -e ".../Besu/bin/besu $GENERAL --bootnodes=$ENODE --p2p-port=30306 --rpc-http-port=8548 $OPTIONS_P2P $OPTIONS_RPC_HTTP"
```

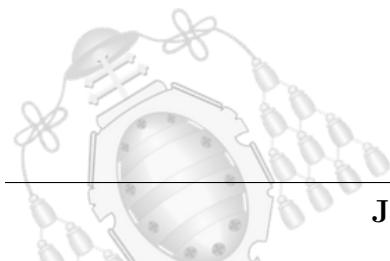
Figura 4.20.- Script para ejecutar los nodos 2, 3 y 4 de la Red B

4.3.4.- Resumen de funcionamiento de la Red B

En este apartado se exponen una serie de capturas que permiten corroborar el buen funcionamiento de esta Red B de prueba creada, así como ver en funcionamiento alguno de los aspectos comentados previamente sobre esta red.

Inicialmente se adjuntan algunas capturas de pantalla utilizando las APIs de JSON-RPC-HTTP. En esas capturas se puede ver diferente información sobre la red y los nodos que la forman. Mediante estas capturas se puede verificar que los cuatro nodos de la red están activos y que todo está funcionando correctamente.

En la figura 4.21 se muestra como se utiliza el método *net_peerCount*. Este método permite obtener el número de nodos conectados que tiene el nodo al que se realiza la petición. En dicha figura se observa como la petición http se realiza cuatro veces, una a cada nodo. Se sabe que se realiza una a cada nodo por el puerto utilizado en la URL, ya que como se ha comentado anteriormente, se utiliza un puerto diferente para cada uno de los nodos. Este puerto se define con la opción *rpc-http-port*. El resultado de la petición es idéntico en los cuatro casos. En todos ellos el resultado es 0x3. Esto indica que todos tienen otros tres nodos conectados.

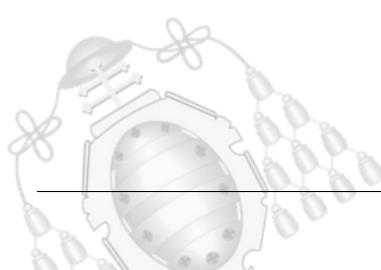




Método utilizado	Nº de Puerto
curl -X POST --data '{"jsonrpc":"2.0", "method":"net_peerCount", "params":[], "id":1}' localhost:8545	
	Resultado en lenguaje hexadecimal
curl -X POST --data '{"jsonrpc":"2.0", "method":"net_peerCount", "params":[], "id":1}' localhost:8546	
	Resultado en lenguaje hexadecimal
curl -X POST --data '{"jsonrpc":"2.0", "method":"net_peerCount", "params":[], "id":1}' localhost:8547	
	Resultado en lenguaje hexadecimal
curl -X POST --data '{"jsonrpc":"2.0", "method":"net_peerCount", "params":[], "id":1}' localhost:8548	
	Resultado en lenguaje hexadecimal

Figura 4.21.- Red B- Número de nodos conectados

El siguiente paso es observar la información de estos nodos. En la figura 4.22 se muestra el uso del método *AdminPeer* que permite obtener la información principal de los nodos que están conectados al nodo al que se le realiza la petición. En este caso, se ha realizado la petición al Nodo1. Esto se sabe porque se utiliza el puerto 8545 que es el puerto utilizado por el Nodo1 para las comunicaciones RPC-HTTP. La información más importante que se puede obtener de cada nodo es el tipo de cliente que utiliza dicho nodo, la dirección IP en la que está escuchando, el identificador del nodo (que es la primera parte de su enode) y también el puerto que utiliza para las comunicaciones P2P. Respecto al puerto, es importante destacar que el número de puerto viene expresado en lenguaje hexadecimal. Por ejemplo para el Nodo2, el valor de puerto que se representa es: 0x7660. Este número, si se pasa a lenguaje decimal, se corresponde con el puerto 30304, que es el que se había suministrado como parámetro en el script mostrado en la figura 4.20.





```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0", "method":"admin_peers", "params":[],"id":1}' http://127.0.0.1:8545
Método utilizado: POST
URL para conectarse al nodo: http://127.0.0.1:8545

{
    "jsonrpc": "2.0",
    "id": 1,
    "result": [
        {
            "version": "0x5",
            "name": "besu/v1.4.1/linux-x86_64/openjdk-java-11",
            "caps": [ "eth/62", "eth/63", "eth/64", "IBF/1" ],
            "network": {
                "localAddress": "192.168.1.35:30303",
                "remoteAddress": "192.168.1.35:54312"
            },
            "port": "0x7662", Puerto utilizado para comunicaciones P2P
            "id": "0xbbe65e98425e5dc07641f3ee1a8f1ca8a161c9077a890155e68debb33429a891ea4a5de27d5e75a182a5ff48e0ecbb47151a71e66ea051fbdd6fec6bdec02f5cc
        },
        {
            "version": "0x5",
            "name": "besu/v1.4.1/linux-x86_64/openjdk-java-11",
            "caps": [ "eth/62", "eth/63", "eth/64", "IBF/1" ],
            "network": {
                "localAddress": "192.168.1.35:30303",
                "remoteAddress": "192.168.1.35:54316"
            },
            "port": "0x7661"
            "id": "0xc90cf19a5d290960a1be5a29515b0a077a5a5dd96dffcf5705dfe48451df99aaef91981cd95355788f32ad2e5c097269cb0a1ca8ab1c64d31d2db3979e2b383
        },
        {
            "version": "0x5",
            "name": "besu/v1.4.1/linux-x86_64/openjdk-java-11",
            "caps": [ "eth/62", "eth/63", "eth/64", "IBF/1" ],
            "network": {
                "localAddress": "192.168.1.35:49814",
                "remoteAddress": "192.168.1.35:30304"
            },
            "port": "0x7660"
            "id": "0xf45979cdcd2355a424df000b28e6df4c402610977298960fe624afbcea768f976fb2676792e53f09677a378116e6070edc029a95b77cb4fdd040f86ea182559
        }
    ]
}
```

Figura 4.22.- Red B- Información sobre los nodos

Otro aspecto interesante es verificar que el identificador de red sea igual para todos los nodos. Este identificador de red debe coincidir con el mismo número que se ha definido en el archivo *genesis.json* mostrado anteriormente. En este caso, el número utilizado para el identificador de red era 160616. Este valor se puede comprobar utilizando el método *net_version*. En este caso se ha utilizado para dos de los nodos activos. El resultado se muestra en la figura 4.23.

```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0", "method":"net_version", "params":[],"id":53}' http://127.0.0.1:8545
Método utilizado: POST
URL para conectarse al nodo: http://127.0.0.1:8545

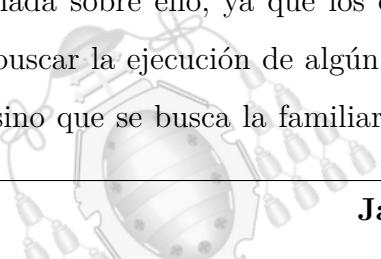
{
    "jsonrpc": "2.0",
    "id": 53,
    "result": "160616" Identificador de red utilizado
}

ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0", "method":"net_version", "params":[],"id":53}' http://127.0.0.1:8546
Método utilizado: POST
URL para conectarse al nodo: http://127.0.0.1:8546

{
    "jsonrpc": "2.0",
    "id": 53,
    "result": "160616"
}
```

Figura 4.23.- Red B- Información sobre Id de la Red

Por otro lado, también es interesante comprobar cómo funciona la red desde el punto de vista de las transacciones que se realizan en la misma. Inicialmente, la red de prueba genera bloques vacíos, ya que no se realiza ninguna transacción ni se ejecuta nada sobre ello, ya que los objetivos del proyecto no contemplan de manera directa buscar la ejecución de algún tipo de aplicación sobre las propias redes de blockchain, sino que se busca la familiarización y la interoperabilidad entre clientes Geth y Besu





en una misma red. De todas formas, realizar ciertas transacciones sobre la red permite verificar que está funcionando correctamente.

Para analizar el funcionamiento de las transacciones, primero, es interesante ver que, cada vez que se añade un nuevo bloque a la cadena, este hecho se muestra por pantalla. Este proceso se procesa en la figura 4.24. Se ve como se completan cada uno de bloques y se añaden a la cadena. Además, en la línea de cada bloque también hay un elemento que indica el número de transacciones que hay en ese bloque. En la mayoría de bloques, hay cero transacciones como se dijo antes, y esto aparece representado como: *0 tx*. En cambio, en el bloque número 2324 el indicador del número de transacciones ha cambiado a 1. Esto es debido a que se ha realizado una prueba sobre esta red. Para realizar esta prueba, se ha utilizado la extensión de navegador conocida como Metamask. Esto permite funcionar a modo de puente entre la red de blockchain y el usuario, de forma que facilite cualquier tipo de movimiento en aplicaciones descentralizadas [52].

```
Round=0), hash=0x0c50d9d632d5279d181aa3930506c99867a2b129c3176c02701c25f7ad557eb
2020-06-18 10:51:29.117+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2322,
Round=0}, hash=0x6c0963a265d03da0080926544dd85364b16ef78e4257b6ccbd8cccd78ed34e64
2020-06-18 10:51:31.346+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2323,
Round=0}, hash=0xf3ebbe55dc34500423ec0d136fe7de990cf57c2a693c7aaa438f02f3b41c8632
2020-06-18 10:51:31.399+02:00 | EthScheduler-Workers-0 | INFO | BlockPropagationManager | Imported #2.323 / 0 tx / 0 om / 0 (0,0%) gas / (0xf
8eb0ee5dc34500423ec0d136fe7de990cf57c2a693c7aaa438f02f3b41c8632) in _0.002s.
2020-06-18 10:51:31.399+02:00 | EthScheduler-Workers-0 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2324,
Round=0}, hash=0x7ca3ed67f99dd7a5b3ce6b4838b68a4c5795e14544548ea9b783e9ec28fd1c2f
2020-06-18 10:51:33.271+02:00 | EthScheduler-Workers-3 | INFO | BlockPropagationManager | Imported #2.324 / 1 tx / 0 om / 21.000 (0,4%) gas /
(0x7ca3ed67f99dd7a5b3ce6b4838b68a4c5795e14544548ea9b783e9ec28fd1c2f) in _0.018s.
2020-06-18 10:51:35.203+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2325,
Round=0}, hash=0x349fce2cc86534c482dca3764571076f9c95f8e085c19f646df3e563e667b709
2020-06-18 10:51:37.277+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2326,
Round=0}, hash=0x03099ffafe9a9dc11b2eb74e7fd9a433f7a798db6b3471cd9a3e82c094b5e20f
2020-06-18 10:51:37.298+02:00 | EthScheduler-Workers-0 | INFO | BlockPropagationManager | Imported #2.326 / 0 tx / 0 om / 0 (0,0%) gas / (0x0
93099ffafe9a9dc11b2eb74e7fd9a433f7a798db6b3471cd9a3e82c094b5e20f) in _0.000s.
2020-06-18 10:51:39.152+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2327,
Round=0}, hash=0xcbe2f3939311d3e432dcb378ef1219bfdae5f554109ae80cc12233720c005e52
2020-06-18 10:51:39.198+02:00 | EthScheduler-Workers-1 | INFO | BlockPropagationManager | Imported #2.327 / 0 tx / 0 om / 0 (0,0%) gas / (0xc
be2f3939311d3e432dcb378ef1219bfdae5f554109ae80cc12233720c005e52) in _0.006s.
2020-06-18 10:51:41.149+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2328,
Round=0}, hash=0xde9e18713731500099d19a9239785bbe4a5fd586a58a62d5a6e61b3fc8622bf
2020-06-18 10:51:41.178+02:00 | EthScheduler-Workers-2 | INFO | BlockPropagationManager | Imported #2.328 / 0 tx / 0 om / 0 (0,0%) gas / (0x0
de9e18713731500099d19a9239785bbe4a5fd586a58a62d5a6e61b3fc8622bf) in _0.000s.
2020-06-18 10:51:43.154+02:00 | pool-10-thread-1 | INFO | IbftRound | Importing block to chain. round=ConsensusRoundIdentifier{Sequence=2329,
Round=0}, hash=0xc6951d01f5fc32aa2dc613f3a605bbfc85917b869eb67db912c9dc5d301cb3db
2020-06-18 10:51:45.555+02:00 | EthScheduler-Workers-3 | INFO | BlockPropagationManager | Imported #2.330 / 0 tx / 0 om / 0 (0,0%) gas / (0xd
742845b6959a31260410e3f4439633bc661aaabb27859bia7344b804e3474b68) in _0.000s.
```

Figura 4.24.- Red B- Nuevos bloques y transacciones

Inicialmente, al entrar en Metamask, hay que definir la red a la que se pretende conectarse. Para realizar las conexiones, Metamask utiliza la configuración de RPC-HTTP utilizada para las APIs de JSON-RPC-HTTP. Por defecto, Metamask tiene ya definidas las redes de blockchain más conocidas (por ejemplo Ethereum). Pero en este caso, lo que se pretende es conectarse a la Red B de prueba recién creada. Para definir esta nueva red en Metamask, es necesario definir el nombre de la red y también la URL



utilizada para realizar la conexión. Esa URL es la misma que se utiliza para conocer a los nodos con las APIs de JSON-RPC-HTTP. La única diferencia que existe respecto a esos casos, es que para eso se utiliza la dirección de loopback (127.0.0.1) porque las peticiones HTTP se realizan desde la misma máquina virtual, pero en este caso, la conexión con Metamask se realiza desde el equipo Anfitrión, por ese motivo no es válida esta dirección. Este proceso de definir la nueva red en Metamask se refleja en la figura 4.25.

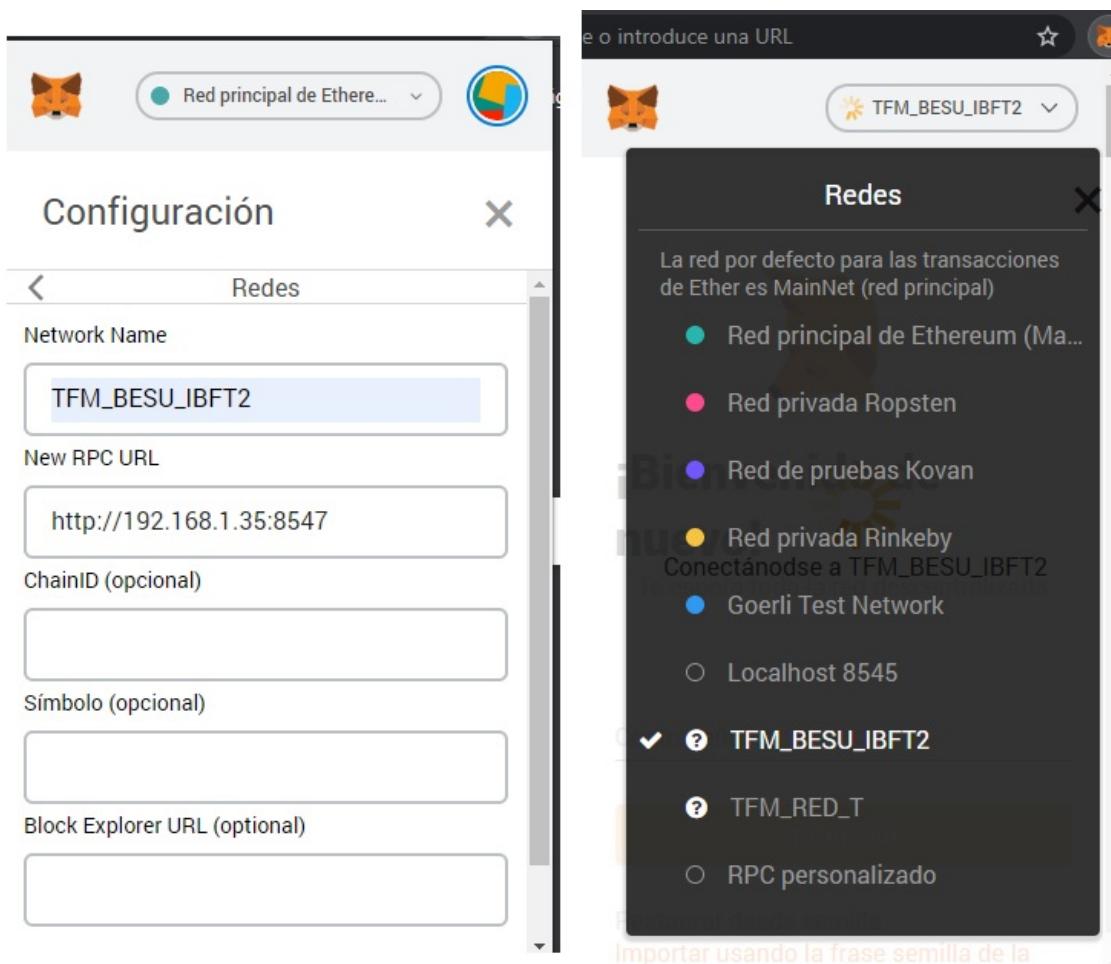
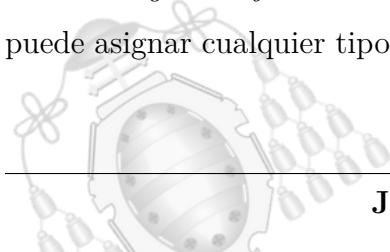


Figura 4.25.- Creación de nueva red en Metamask

Una vez se ha realizado la conexión a la red, ya se puede realizar la transacción. Para realizar la transacción se usan las cuentas ficticias que han sido predefinidas en el archivo *genesis.json*. Al tratarse de una red de prueba, a las cuentas usadas se les puede asignar cualquier tipo de saldo, ya que son valores ficticios. En la figura 4.26, se





muestra el historial de transacciones, visto desde Metamask, y que demuestra que se ha realizado una transacción.

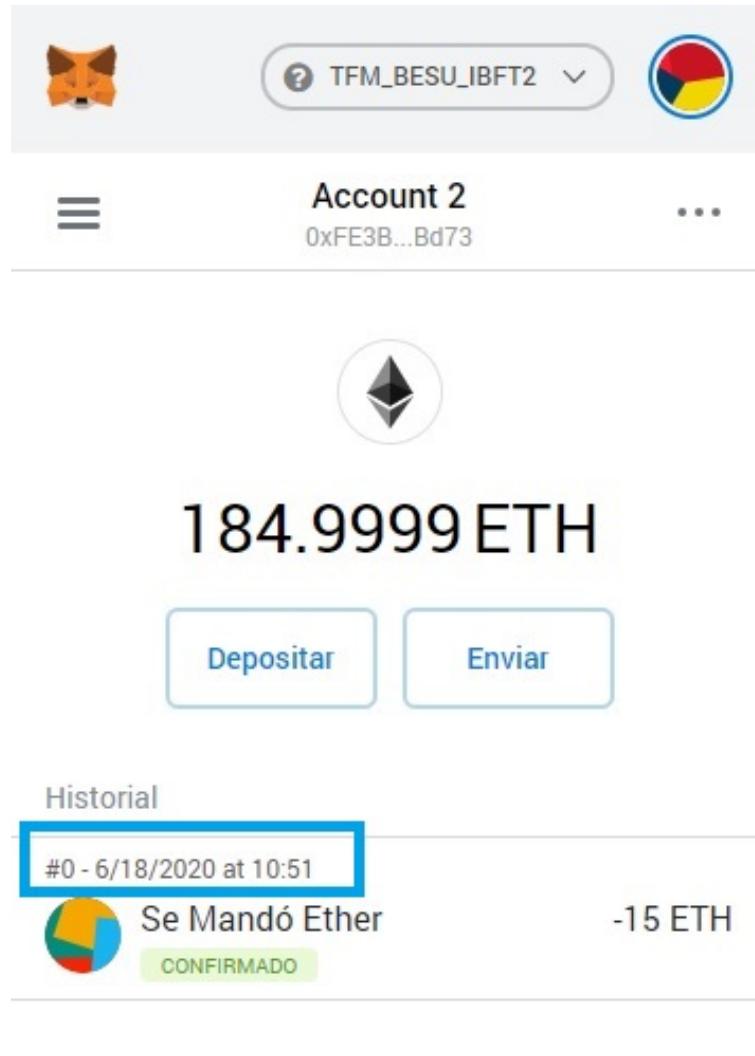
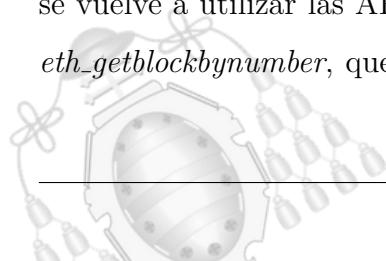


Figura 4.26.- Historial de transacciones desde Metamask

Si se observa el historial de transacciones, la fecha y hora de la transacción realizada coincide exactamente con la fecha y hora del bloque (número 2324) mostrado en la figura 4.24 que contiene una transacción.

Por otro lado, también se puede comprobar cómo es la información generada por una transacción desde el punto de la red de blockchain. Para consultar esta información, se vuelve a utilizar las APIs de JSON-RPC-HTTP. En este caso, se utiliza el método `eth_getblockbynumber`, que permite obtener la información contenida en un bloque en





concreto. Es importante destacar que el número de bloque que se pretende analizar tiene que escribirse en formato hexadecimal a la hora de realizar la petición HTTP.

En la figura 4.27 se muestra la información asociada al bloque 2324. Se ve como en el campo de transacciones sí que hay registrada cierta información asociada a la transacción mostrada anteriormente. En cambio, en la figura 4.28 el campo de transacciones se encuentra vacío. Esto se debe a que esa figura se corresponde a otro bloque, y como ya se ha visto, el resto de bloques están vacíos.

```
ubuntu1@ubuntul1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getBlockByNumber", "params": ["0x914"], "true}, "id":1}' http://127.0.0.1:8545
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : {
    "number" : "0x914", Nº de bloque
    "hash" : "0x7ca3ed6f99dd7a5b3ce6b4838b68a4c5795e14544548ea9b783e9ec28fd1c2f", Método utilizado para obtener la información
    "transactions" : [ { Campo transacciones, en el que se incluye la información de las transacciones contenidas en ese bloque
      "blockHash" : "0x7ca3ed6f99dd7a5b3ce6b4838b68a4c5795e14544548ea9b783e9ec28fd1c2f",
      "blockNumber" : "0x914",
      "from" : "0xfe3b557e8fb62b89f4916b721be55ceb828bdb73",
      "gas" : "0x5208",
      "gasPrice" : "0x165a0bc00",
      "hash" : "0x9c3340c38e3c2326a72d681789992fc80b18c95556e81430c6f7d26bda2db18",
      "input" : "0x",
      "nonce" : "0x0",
      "to" : "0x1e10c40eee4a2420ccd455fc1933966ec2cc389b",
      "transactionIndex" : "0x0",
      "value" : "0xd02ab486cedc0000",
      "v" : "0x46f4",
      "r" : "0xe39f9b5127e24451bbcc046494432832b868c3f0c0f0587af53da289b928c4e9",
      "s" : "0x67d5b8a752f9dde778027e5dcc82d96b5bc8f75d4b827874724296fba7198fa"
    } ]
}

```

Figura 4.27.- Información del bloque que contiene una transacción

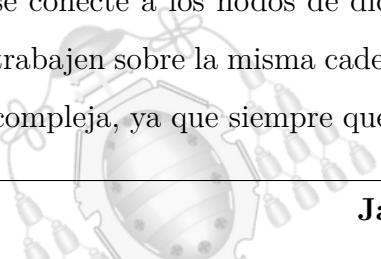
```
ubuntu1@ubuntul1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0", "method": "eth_getBlockByNumber", "params": ["0x915"], "true}, "id":1}' http://127.0.0.1:8545
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : {
    "number" : "0x915",
    "transactions" : [ ] En este caso el campo reservado para las transacciones está vacío,
    este bloque no contiene ninguna transacción
  }
}
```

Figura 4.28.- Información de un bloque que no contiene transacciones

4.4.- Interoperabilidad entre cliente Geth y Besu

Finalmente, se plantea la estrategia utilizada para investigar la forma en la que se pueda lograr la interoperabilidad entre estos dos tipos de clientes.

El objetivo que se pretende conseguir es que a partir de una Red T, similar a la utilizada anteriormente, hacer que un nodo, implementado mediante el cliente Besu, se conecte a los nodos de dicha Red T y se pueda sincronizar con ellos de forma que trabajen sobre la misma cadena de bloques. A simple vista, se presenta como una tarea compleja, ya que siempre que se trabaja con tecnologías diferentes, es evidente que las





formas de implementación, la configuración y su manera de funcionamiento difiere. Por dicho motivo, a partir de lo aprendido en la implementación de las redes de prueba, se intenta definir y modificar todos los parámetros necesarios de ambos clientes para que ambas tecnologías puedan funcionar de manera homogénea.

4.4.1.- Parámetros fundamentales a considerar

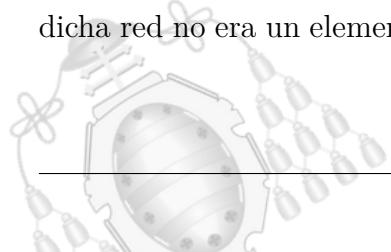
En este apartado se detallan los elementos fundamentales que se deben tener en cuenta para intentar lograr el objetivo final del proyecto, es decir, buscar la interoperabilidad entre los clientes Geth y Besu en una misma red de blockchain.

4.4.1.1.- Análisis del origen de la red

El primer elemento que se debe considerar si queremos conseguir la interoperabilidad de los dos clientes es el origen del bloque cero de la red que forman los nodos implementados mediante cada cliente. En el caso de las redes de prueba, no se ha prestado especial atención al origen de la cadena de bloques, debido a que cada red se ha definido de manera independiente y todos los nodos de cada red siguiendo la misma metodología y utilizando los mismos ficheros de configuración.

Pero en este nuevo escenario, los nodos son creados utilizando diferentes tecnologías, por dicho motivo, si no se configura adecuadamente el origen de la cadena de bloques, los nodos no pueden conectarse entre ellos, ya que dos nodos que no comparten el mismo origen de red no pueden sincronizarse debido a que se generan incompatibilidades en la cadena.

Cuando se habla del origen de la cadena de bloques se debe pensar en el archivo génesis. Este archivo se utiliza para definir los parámetros de configuración de la cadena de bloques y por tanto definir el bloque cero o bloque génesis de la cadena. En el caso de la Red T de prueba definida anteriormente, no se ha utilizado un archivo génesis porque se utiliza la configuración de la red por defecto, ya que para la fase de análisis de dicha red no era un elemento destacable. En cambio, en el caso de la Red B de prueba,





sí que se ha utilizado el archivo génesis para definir la configuración de la cadena de bloques.

Para este nuevo escenario es imprescindible crear un archivo génesis que permita definir las características básicas de la nueva red homogénea y que sean válidas para los dos tipos de clientes. La importancia de este archivo génesis no reside en los propios parámetros de configuración que se definen en él, sino que reside en que es la forma de garantizar que los nodos definidos mediante ambos clientes utilizan el mismo bloque cero de la cadena y por tanto, no se produzcan incompatibilidades.

Recordando lo mencionado en apartados anteriores, los archivos génesis tienen cuatro campos obligatorios. Estos campos son:

- **Config:** En este caso es el campo al que más atención se le presta, ya que dentro de este campo se define el valor del identificador de la red (Networkid) y este valor también tiene gran relevancia en este caso.
- **Alloc:** Este campo sirve para predefinir cuentas y balances en cada nodo. En este caso, el contenido de este campo no es de importancia para lograr el objetivo del proyecto.
- **GasLimit:** Utilizado para delimitar el Gas que se puede gastar en generar un bloque. El valor que adquiere este parámetro tampoco es determinante en este caso.
- **Difficulty:** Indica la dificultad para añadir nuevos bloques a la cadena. Cuanto más alto sea el valor, más difícil es añadir nuevos bloques.

4.4.1.2.- Identificador de la red

Otro de los elementos imprescindibles para poder lograr el objetivo deseado es el identificador que tienen todas las cadenas de bloques. Es fundamental que este identificador numérico sea el mismo en el cliente Geth que en el cliente Besu, ya que en caso de ser diferente, es imposible que se puedan conectar ambos a la misma red.

Este parámetro, en el caso del cliente Besu es definido en el archivo génesis solamente, y en el caso del cliente Geth se define en el archivo génesis, pero también se



define en la línea de comandos. Este identificador puede ser cualquier número entero positivo diferente de 1, 3, 4, 5, 6, 53, 61 y 2018 ya que estos son números asignados a las redes públicas de blockchain o a otras redes de prueba ya definidas a nivel global. En caso de utilizar uno de estos identificadores, es posible conectarse a una de estas redes, pero esto no es parte del objetivo del proyecto, y por tanto se trabaja con otro número entero positivo diferente a ese. En este caso se utiliza el número: 160616.

4.4.1.3.- Permisionado de la red

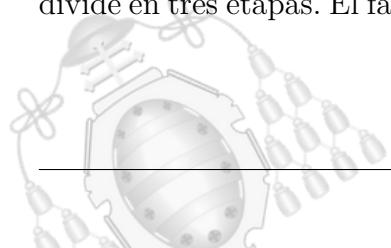
El tercer elemento que se debe tener en cuenta para buscar la interoperabilidad entre clientes es el permisionado que se le aplica a la red. Tanto el cliente Geth como el cliente Besu ofrecen la opción de aplicar permisionado. En este caso, al tratar de buscar un escenario lo más cercano posible a las Redes de Alastria originales, se añade el permisionado a la red. Es decir, solo unos nodos concretos pueden conectarse a la Red. Para que los nodos puedan conectarse a una red con permisionado activo, el enode de estos nodos tiene que estar registrado en el fichero correspondiente. Generalmente, este fichero recibe permissioned-nodes.json y su efecto puede ser aplicado a nivel de toda la red o de manera individual para cada nodo.

4.4.1.4.- Conexión entre máquinas virtuales

Este último elemento que se describe no está ligado con la tecnología de blockchain como tal, si no que está relacionado con cualquier tipo de programa que requiere comunicación entre elementos que se encuentran en dos máquinas diferentes. Es imprescindible que las máquinas virtuales tengan conexión entre ellas. De lo contrario es imposible que los clientes se consigan conectar.

4.4.2.- Procedimiento seguido

El procedimiento utilizado para alcanzar la interoperabilidad entre los clientes se divide en tres etapas. El factor común a las tres etapas es el número de nodos utilizados.





En todas ellas se utilizan tres nodos. Dos implementados mediante el cliente Geth y uno implementado mediante el cliente Besu. A continuación se definen estas tres etapas:

- La primera etapa consiste en diseñar una red sin permisionado, en la que utilizando un bootnode se conecten los dos nodos de Geth con el nodo implementado con Besu. En la figura 4.29 se muestra un esquema de la arquitectura de red planteada.
- La segunda etapa consiste en utilizar la misma red que en la etapa uno, pero añadiendo permisionado.
- La tercera etapa se basa en la integración del cliente Besu, en una Red T de prueba formada por el nodo Main y un nodo validador.

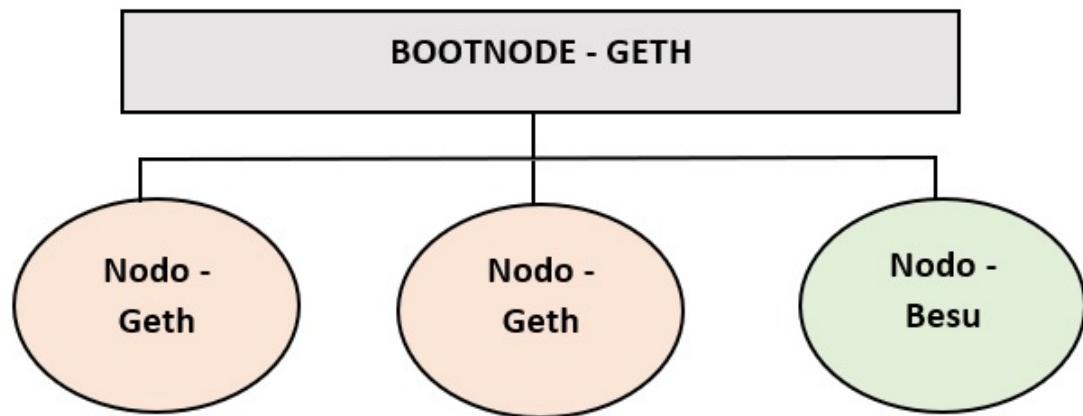


Figura 4.29.- Arquitectura de la red utilizada

4.4.2.1.- Conexión de nodos sin permisionado

La primera aproximación al objetivo del proyecto, se ha realizado en una red sin permisionado, es decir, cualquier nodo conectado al bootnode puede conectarse con otros nodos conectados al mismo bootnode, siempre considerando que cumplan los requisitos del archivo génesis y del identificador de red.

El primer paso que se realiza es poner en funcionamiento el bootnode, para ello se ha seguido las instrucciones dadas en la referencia [53]. Primero se genera la clave necesaria para conseguir a partir de ella el valor del enode del bootnode. Para generar esa clave automáticamente se usa uno de los comandos mencionados en [53] y que se



muestra en la figura 4.30. Una vez esta clave es creada y almacenada en un fichero, se inicia el bootnode pasándole como parámetro el fichero que contiene la clave. Una vez se ha ejecutado el bootnode, aparece por consola el valor del enode. Este valor es esencial para poder ejecutar el resto de nodos que forman la red, ya que es necesario pasárselo como parámetro para que se puedan conectar a él. Todo este procedimiento de ejecución del bootnode se ve en la figura 4.30.

```
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/Ejemplo$ bootnode --genkey=clave.key Genera la clave y la almacena en el fichero clave.key
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/Ejemplo$ cat clave.key
5972df58b59269fe4ce900ecd7fdcce520214af1130a182b71a8dc16360579fubuntu1@ubuntu1-VirtualBox:~/Escritorio/Ejemplo$ 
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/Ejemplo$ bootnode --nodekey clave.key Ejecuta el bootnode utilizando la clave creada
INFO [06-28|09:43:52] UDP listener up self=enode://780dd51080f5771d0b1cbe06a5db8fc1ac27109b9a518e1d0e82545292f90e38f17a5f48c91281cbd3170a1cdc25f33514b5a55ba33f7a1de9699857d2e82ae5[::]:30301
```

Figura 4.30.- Puesta en funcionamiento del bootnode

Una vez que se ha iniciado el bootnode, se empieza con la iniciación del resto de nodos de la red. Primero, se pretende realizar la conexión entre los dos nodos implementados mediante el cliente Geth, qué están ejecutándose sobre la misma máquina virtual. Para facilitar el proceso de ejecución de los nodos implementados mediante el cliente Geth, y ahorrar tiempo en las diferentes pruebas realizadas, se automatiza el proceso de puesta en marcha de los nodos. La automatización se realiza mediante la programación de unos scripts en Bash. Estos scripts son como el mostrado en la figura 4.31 y se utiliza uno para cada uno de los nodos Geth. En ambos casos, lo único que cambia de un script a otro es el directorio utilizado para almacenar el contenido de la cadena de bloques y también alguna de las opciones de la línea de comandos del Geth.

```
#!/bin/bash

#Este archivo recibe como parametro el ID del bootnode... $1

ID=$1
touch data/IP.txt data/IP2.txt data/IP3.txt
ifconfig > data/IP.txt
head -2 data/IP.txt | tee data/IP2.txt
tail -1 data/IP2.txt | tee data/IP3.txt
IP=$(cut -d' ' -f 10 data/IP3.txt)

rm -rf data
mkdir data

ENODE="enode://${ID}@${IP}:30301"

geth --datadir $(pwd)/data init genesis.json Definición del bloque cero en el directorio elegido
geth --datadir $(pwd)/data --networkid=160616 --bootnodes "$ENODE" --rpc --rpcport=8546 Inicio del cliente Geth
```

Figura 4.31.- Script para automatización de los nodos del cliente Geth



La primera parte de este script es utilizada para obtener la dirección IP de la máquina virtual en la que se ejecuta el bootnode y los dos nodos de Geth. Esta dirección IP es almacenada en una variable dentro del script. Posteriormente, esta variable es utilizada para formar el enode del bootnode. Es importante recordar que los enodes están formados por: ID_Nodo@Dirección_IP:Puerto. En este caso, el valor del puerto se escribe manualmente ya que siempre es el mismo. Se utiliza el puerto 30301, y el valor del ID_Nodo se recibe como parámetro a la hora de ejecutar el script.

La segunda parte del script es la más importante y es la que está relacionada directamente con el uso del cliente Geth. Lo primero que se realiza es elegir el directorio donde se almacena la cadena de bloques y en ese directorio, mediante el comando *init* se fija el origen de la cadena de bloques a partir de la configuración existen en el archivo *genesis.json*. Finalmente, se ejecuta el nodo, utilizando como bloque cero el creado a partir del archivo génesis. Además, a la hora de invocar el nodo, se le pasan como parámetro algunas opciones que son requeridas para el correcto funcionamiento. Estas opciones son:

- **Bootnode:** Esta opción es imprescindible, ya que indica al nodo el valor del enode del bootnode al que tiene que conectarse.
- **Networkid:** Este parámetro permite definir el identificador de red utilizado.
- **Port:** En el caso del segundo nodo que se ejecuta, se utiliza esta opción para definir el puerto utilizado para las comunicaciones P2P. Para el Nodo1 se utiliza el valor por defecto que es 30303.

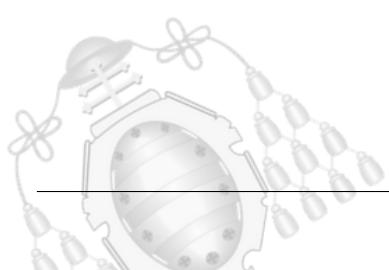
A continuación se muestra una figura con el contenido del archivo génesis utilizado en este caso. Se debe tener en cuenta que se utilizan valores aleatorios, sin ningún criterio en concreto, ya que el objetivo del proyecto es conseguir la interoperabilidad entre las dos tipos de clientes, y para eso únicamente se requiere que la información que el archivo génesis sea detectado por los clientes a la hora de iniciar el nodo, por tanto, no es necesaria una configuración determinada de ciertos parámetros. Se toma el valor del campo *difficulty* como valor de referencia para garantizar que los nodos están utilizando adecuadamente el archivo génesis. Se configura el valor de 0x10 a este campo. Este valor convertido a lenguaje decimal se corresponde con 16.

Figura 4.32.- Genesis.json

Una vez que se han ejecutado los dos nodos Geth, es importante comprobar que ambos se detectan entre ellos y están conectados entre sí. Para verificar esta información se accede a la consola de Javascript de cada uno de ellos, de forma que se utiliza alguno de los comandos ya vistos anteriormente, para obtener la información sobre los nodos que tienen directamente conectados.

En la figura 4.33 se muestra la información sobre el primero de los nodos Geth y en la figura 4.34 la información relativa al segundo de los nodos. De estas dos figuras se deben destacar los siguientes datos:

- El valor de los enode de cada uno de los nodos, ya que posteriormente permiten verificar cómo están conectados entre ellos.
 - El campo *listenaddress* que es el campo que define el puerto por donde escucha el nodo las comunicaciones P2P. El primero de los nodos escucha en el puerto 30303 y el segundo en el 30305.
 - El campo de *difficulty* muestra el valor 16.
 - El campo *network* muestra el valor 160616 que es el valor fijado como identificador de red.
 - El campo *genesis* es igual en ambos nodos. Esto indica que los dos utilizan el mismo origen de la cadena de bloques.





```
> admin.nodeInfo
{
  enode: "enode://d2e2b2489f5f5331a9c76d3196aa6c9276cf24714971fe990a27c7796297898a9519edc9348db75dd4cbaf45aa7d4c31bd52e5a601ed9dc9599d296ff2ee1df4@92.185.221.150:30303",
  id: "d2e2b2489f5f5331a9c76d3196aa6c9276cf24714971fe990a27c7796297898a9519edc9348db75dd4cbaf45aa7d4c31bd52e5a601ed9dc9599d296ff2ee1df4",
  ip: "92.185.221.150",
  listenAddr: "[::]:30303",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  ports: {
    discovery: 30303,
    listener: 30303
  },
  protocols: {
    eth: {
      difficulty: 16,
      genesis: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      head: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      network: 160616
    }
  }
}
```

El campo dificultad tiene el valor definido en el genesis.json
El valor del campo genesis es igual para los dos nodos. Mismo bloque cero.
El valor del identificador de red es el definido en el genesis.json

Figura 4.33.- Información del nodo 1 implementado con Geth

```
> admin.nodeInfo
{
  enode: "enode://1c69785ddc66b29ac8b087d507e0f3a13acdeb350b98132b5c395af8575dc73567789f06c00f6212c839cd1894dfb7d75d2d90bd428b9aed8e1a0abe08c7ee21@92.185.221.150:30305",
  id: "1c69785ddc66b29ac8b087d507e0f3a13acdeb350b98132b5c395af8575dc73567789f06c00f6212c839cd1894dfb7d75d2d90bd428b9aed8e1a0abe08c7ee21",
  ip: "92.185.221.150",
  listenAddr: "[::]:30305",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  ports: {
    discovery: 30305,
    listener: 30305
  },
  protocols: {
    eth: {
      difficulty: 16,
      genesis: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      head: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      network: 160616
    }
  }
}
```

Figura 4.34.- Información del nodo 2 implementado con Geth

En la figura 4.35 se muestra la información que presenta cada uno de los dos nodos sobre el resto de nodos que tienen conectados. En este caso, se ve como cada uno de los nodos tiene conectado el otro nodo. Esto se puede comprobar observando el valor de los enode de cada uno de los nodos, ya que se corresponden en cada caso con los mostrados en las figuras 4.33 y 4.34.

```
> admin.peers
[{
  caps: ["eth/63"],           Valor del ID del nodo que tiene conectado
  id: "1c69785ddc66b29ac8b087d507e0f3a13acdeb350b98132b5c395af8575dc73567789f06c00f6212c839cd1894dfb7d75d2d90bd428b9aed8e1a0abe08c7ee21",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:50436",
    remoteAddress: "192.168.1.83:30305"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      version: 63
    }
  }
}
> net.version
"160616"                   Identificador de la red a la que está conectado
> net.peerCount
1                           Número de nodos que tiene conectados
```



```
> admin.peers
[{
  caps: ["eth/63"],           Valor del ID del nodo que tiene conectado
  id: "d2e2b2489f5f5331a9c76d3196aa6c9276cf24714971fe990a27c7796297898a9519edc9348db75dd4cbaf45aa7d4c31bd52e5a601ed9dc9599d296ff2ee1df4",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:30305",
    remoteAddress: "192.168.1.83:50436"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xaax469570507706e6d7e0815ceea0eea9a4b76cb5bad47b03a8329856aa87a3b1",
      version: 63
    }
  }
}
> net.version
"160616"                   Identificador de la red a la que está conectado
> net.peerCount
1                           Número de nodos que tiene conectados
```

Figura 4.35.- Información sobre los nodos conectados



A partir de este punto, se pasa a trabajar con el cliente Besu. Al igual que se ha realizado con los nodos implementados mediante el cliente Geth, se trata de automatizar la ejecución del nodo implementado mediante el cliente Besu a partir de un script programado en Bash.

En la primera parte de este script se definen variables con diferentes opciones de la línea de comandos que son utilizadas a la hora de ejecutar el nodo. Principalmente, están relacionadas con las opciones habilitadas para las comunicaciones RPC y con la configuración de la red. Parte de estas opciones de configuración incluyen el archivo génesis utilizado. Este archivo es igual al mostrado en la figura 4.32 y que se utiliza con los otros dos nodos implementados mediante el cliente Geth.

La segunda parte de este script está relacionada con la ejecución del cliente Besu. En este caso, simplemente se le pasan como parámetro las variables definidas con las opciones de configuración y también se le pasa como otro parámetro el valor del enode del bootnode. Es imprescindible pasarle el valor del bootnode, ya que de lo contrario no se puede realizar la conexión con los otros dos nodos. Este script se muestra en la figura 4.36.

```
#!/bin/bash
# -*- ENCODING: UTF-8 -*-

OPTIONS_P2P="--p2p-enabled=true --p2p-host=192.168.1.35 --p2p-interface=0.0.0.0"
OPTIONS_RPC_HTTP="--rpc-http-enabled --rpc-http-host=0.0.0.0 --rpc-http-api=ETH,NET,IBFT,ADMIN --rpc-http-cors-origins='all'"
GENERAL="--data-path=data --genesis-file=../genesis.json --host-whitelist='*'"

ENODE=$1
cd Node2
./../Besu/bin/besu $GENERAL --bootnodes=$ENODE --p2p-port=30304 --rpc-http-port=8546 $OPTIONS_P2P $OPTIONS_RPC_HTTP
```

Figura 4.36.- Script para automatización del nodo del cliente Besu

Finalmente, tras poner en funcionamiento los nodos que forman la arquitectura de la red, se comprueba si se ha conseguido establecer la conectividad entre ellos, de forma que, de cara al futuro, se puedan implementar en una red única y homogénea.

En la figura 4.37 se muestra la información del nodo de Besu. Para conseguir esta información se ha utilizado la API de JSON-RPC-HTTP. En la información mostrada se destaca como el campo *difficulty* es 16, al igual que los nodos anteriores. Otro



parámetro que se debe resaltar es el identificador de la red, ya que también vuelve a coincidir con el valor mostrado en los casos anteriores, 160616.

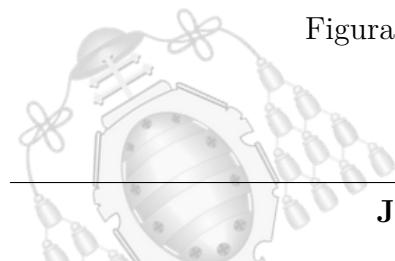
```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"admin_nodeInfo","params":[],"id":1}' http://127.0.0.1:8546
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : {
    "enode" : "enode://b3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3baf9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa@192.168.1.35:30304",
    "listenAddr" : "192.168.1.35:30304",
    "ip" : "192.168.1.35",
    "name" : "besu/v1.4.1/linux-x86_64/openjdk-java-11",
    "id" : "b3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3baf9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa",
    "ports" : [
      "discovery" : 30304,
      "listener" : 30304
    ],
    "protocols" : {
      "eth" : {
        "config" : {
          "chainId" : 160616,
          "constantinopleFixBlock" : 0,
          "ibft2" : {
            "epochLength" : 30000,
            "blockPeriodSeconds" : 2,
            "requestTimeoutSeconds" : 10
          }
        },
        "difficulty" : 16,
        "genesis" : "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
        "head" : "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
        "network" : 160616
      }
    }
  }
}
```

Figura 4.37.- Información del nodo implementando con el cliente Besu

Finalmente se comprueba si se ha conseguido que los tres nodos se conecten y se sincronicen entre ellos. Para verificar esto, se muestra la información de los nodos conectados a cada uno de los tres. Para que todo este correcto, cada uno de los nodos debe tener dos nodos como vecinos. Para conseguir esta información, en el nodo de Besu se utiliza nuevamente la API de JSON-RPC-HTTP y en los nodos de Geth se utiliza la consola de Javascript.

```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":"admin_peers","params":[],"id":1}' http://127.0.0.1:8546
{
  "jsonrpc" : "2.0",
  "id" : 1,
  "result" : [
    {
      "version" : "0x5",
      "name" : "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
      "caps" : [ "eth/63" ],
      "network" : {
        "localAddress" : "192.168.1.35:33932",
        "remoteAddress" : "192.168.1.83:30305"
      },
      "port" : "0x0",
      "id" : "0x455924ec0092369af70cf52a6b323e4934d1f8dab56be1f221b0efe825f19130f2259efe1403e914b868fb6d7f1acc9c93f18ac6f4357e71d946dd47836b2919",
      "version" : "0x5",
      "name" : "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
      "caps" : [ "eth/63" ],
      "network" : {
        "localAddress" : "192.168.1.35:58428",
        "remoteAddress" : "192.168.1.83:30303"
      },
      "port" : "0x0",
      "id" : "0xb6ad0a60ae163104a78fc6001d3a488141d29b8075d44cd455dd9f05967a7d305d5f9844aa39fc35364123ef862fe8898fd5eb24cf31983ae5735a2e272051e8
    }
  ]
}
```

Figura 4.38.- Nodos vecinos del nodo Besu





En la figura 4.38 se muestra el resultado de la petición HTTP. Se ve como el nodo de Besu está conectado a otros dos nodos, que son los nodos Geth anteriormente implementados. Por otro lado, también se debe comprobar que desde el punto de vista de los nodos Geth, se ha conseguido la conectividad con el nodo implementado con el cliente Besu. Estos resultados se muestran en las figuras 4.39 y 4.40.

```
> admin.peers
[{
  caps: ["IBF/1", "eth/62", "eth/63", "eth/64"],
  id: "b3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3ba9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa",
  name: "besu/v1.4.1/linux-x86_64/openjdk-java-11",
  network: {
    localAddress: "192.168.1.83:30305",
    remoteAddress: "192.168.1.35:33932"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}, {
  caps: ["eth/63"],
  id: "b6ad0a60ae163104a78fc0001d3a488141d29b8075d44cd455dd9f05967a7d305d5f9844aa39fc35364123ef862fe8898fd5eb24cf31983ae5735a2e272051e8",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:34950",
    remoteAddress: "192.168.1.83:30303"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}]
```

Figura 4.39.- Nodos vecinos de uno de los nodos Geth

```
> admin.peers
[{
  caps: ["eth/63"],
  id: "455924ec0092369af70cf52adb323a4934df8dab56he1f221hqeFe825f19130f2259efe1403e914b868fb6d7f1acc9c93f18ac6f4357e71d946dd47836b2919",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:30303",
    remoteAddress: "192.168.1.83:34950"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}, {
  caps: ["IBF/1", "eth/62", "eth/63", "eth/64"],
  id: "b3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3ba9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa",
  name: "besu/v1.4.1/linux-x86_64/openjdk-java-11",
  network: {
    localAddress: "192.168.1.83:30303",
    remoteAddress: "192.168.1.35:58428"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}]
```

Figura 4.40.- Nodos vecinos del otro de los nodos Geth

En estas dos figuras se observa qué cada nodo implementado con el cliente Geth tiene como nodos vecinos a un nodo basado en un cliente Geth y otro basado en un cliente Besu. Otro elemento destacable en esa figura es que todos tienen el mismo



campo *genesis*. Este campo hace referencia al Hash del bloque cero. En los tres casos es igual ya que los tres clientes utilizan el mismo archivo génesis para definir el origen de la cadena.

4.4.2.2.- Conexión de nodos con permisionado

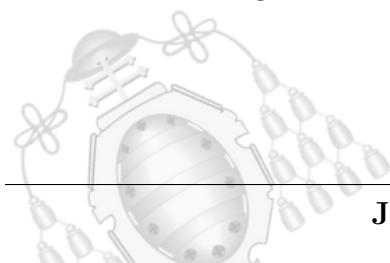
En este apartado se plantea el mismo tipo de red que el caso anterior pero añadiéndole la opción de permisionado. De esta forma, el planteamiento de red utilizado es más próximo a lo que ofrece Alastria en realidad.

Las redes que utilizan permisionado necesitan tener el archivo *permissioned-nodes.json*. En este archivo se incluyen los enodes de los nodos que se permiten en la red. De esta forma, solo los nodos que tienen su enode registrado en el archivo pueden conectarse a la red.

Como punto de partida se toma la red utilizada para el caso anterior, pero ahora, se le añade la opción *-permissioned* a uno de los nodos Geth en el momento de su ejecución. Esta opción indica que para poder conectarse a ese nodo, el enode del nodo que quiere conectarse debe estar incluido en el fichero *permissioned-nodes.json*. Inicialmente, si no se incluye ningún enode dentro de este archivo, el nodo que tiene activado el permisionado no puede encontrar ningún nodo vecino. Esto se refleja en la figura 4.41.

```
> admin.nodeInfo
{
  enode: "enode://f69d4b950efa3ccb94124196b98de2f60f28554057f7de69b257c1f81e3c691d6264f67628193696b74a1b8c202e950d9916a2e6085cb3f7d5434f91291a29f3",
  id: "f69d4b950efa3ccb94124196b98de2f60f28554057f7de69b257c1f81e3c691d6264f67628193696b74a1b8c202e950d9916a2e6085cb3f7d5434f91291a29f3",
  ip: "92.185.221.150:30305",
  listenAddr: "[::]:30305",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  ports: {
    discovery: 30305,
    listener: 30305
  },
  protocols: {
    eth: {
      difficulty: 16,
      genesis: "0xe1278dc7798427ddebfbe39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1",
      head: "0xe1278dc7798427ddebfbe39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1",
      network: 160616
    }
  }
}
> admin.peers
[]
```

Figura 4.41.- Nodos Geth sin encontrar nodos vecinos





Para solucionar el problema mostrado en 4.41, se añade el enode del otro nodo Geth en el archivo *permissioned-nodes.json*. Esto se muestra en la figura 4.42.

```
ubuntu1@ubuntui-VirtualBox:~/Escritorio/Permissionado$ cat permissioned-nodes.json
[{"enode://f69d4b950efa3ccb94124196b98de2f60f28554057f7de69b257c1f81e3c691d6264f67628193696b74a1b8c202e950d9916a2e6085cb3f7d5434f91291a29f3@92.185.221.150:30305"}]
```

Figura 4.42.- Archivo *permissioned-nodes.json* con un enode añadido

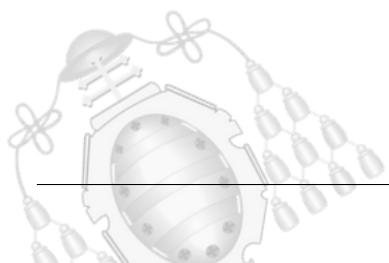
Una vez incluido el valor de ese enode en el archivo correspondiente, se comprueba de nuevo si el nodo que tiene el permisionado activo ha conseguido conectarse a algún nodo ahora que un enode ha sido añadido al archivo de permisionado. Esto se muestra en la figura 4.43.

```
> admin.peers
[{"enode": "enode://63", "id": "695feb25e75084cc8a5c63758420e747ba8276b9f0627cd631119d0412c31ec87b17e9c7199c9290ac5131c97ae769e0c8325419ddbf4f3fdadb2d7532ee53", "name": "Geth/v1.7.2-stable-94e1e31e/Ubuntu-AMD64/go1.10.4", "network": {"localAddress": "192.168.1.83:30305", "remoteAddress": "192.168.1.83:43314"}, "protocols": {"eth": {"difficulty": 16, "head": "0xe1278dc7798427ddebfbef39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1", "version": 63}}}]
```

Figura 4.43.- Nodo conectado incluido en el permisionado

En esta figura se muestra como el nodo con el enode añadido en el archivo *permissioned-nodes.json*, mostrado en la figura 4.42, aparece como un nodo vecino del nodo que tiene el permisionado activo. Esto demuestra que las opciones de permisionado implementadas están funcionando correctamente, ya que cuando se ha añadido el enode al archivo de *permissioned-nodes.json*, se ha podido realizar la conexión entre ambos nodos.

Ahora se realiza el mismo proceso para el nodo implementado con Besu. Primero, se añade el enode de este nodo al archivo de permisionado. Entonces, el archivo *permissioned-nodes.json* tiene el aspecto mostrado en la figura 4.44.





```
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/Permisionado$ cat permissioned-nodes.json
[{"enode": "0xb3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3baf9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa@192.168.1.35:30304", "enode": "0x9edb14e14f7838f523c70126995b9e61c0087c494e0bf5253bf2df5ab2216b43bf880413c240ccf3a5fb2a31873b7164e3064806eb13da7b8389dec05abcf7b6@92.185.221.150:30305"}]
```

Figura 4.44.- Archivo *permissioned-nodes.json* con dos enodes añadidos

Se pretende verificar que se cumple la condición de permisionado con el nodo implementado mediante el cliente Besu y por tanto, se conecta con el nodo del cliente Geth que tiene la opción de permisionado activa. En la figura 4.45 se muestra el resultado de esto. Los nodos se han conectado correctamente. Esto se ve observando los enodes remarcados en la figura. Estos enodes se corresponden con los mostrados en la figura 4.44.

```
> admin.peers
[{
  caps: ["eth/63"],
  id: "9edb14e14f7838f523c70126995b9e61c0087c494e0bf5253bf2df5ab2216b43bf880413c240ccf3a5fb2a31873b7164e3064806eb13da7b8389dec05abcf7b6",
  name: "Geth/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:44506",
    remoteAddress: "192.168.1.83:30305"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}, {
  caps: ["TRE/1", "eth/62", "eth/63", "eth/64"],
  id: "b3b252fffc81bc43cc205a415aa584949ed2f15a8ceccb3baf9fef40137daf0d200e976ab72fd4574d7353b654223cc6943052bd5128195e560b0a1f773fe6fa",
  name: "besu/v1.4.1/linux-x86_64/openjdk-java-11",
  network: {
    localAddress: "192.168.1.83:30303",
    remoteAddress: "192.168.1.35:42030"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xc8959681ec3ac4447b6fbe88172dab17793fb2f5adef732b65a378918817ce8d",
      version: 63
    }
  }
}]
```

Figura 4.45.- Nodos conectados al nodo que tiene el permisionado activo

Finalmente, se ha conseguido la interoperabilidad entre los clientes Geth y Besu en una red que incorpora opciones de permisionado. Esto permite acercarse cada vez más al concepto ofrecido por Alastria de red permisionada.

4.4.2.3.- Integración de nodo Besu en la Red T de Alastria

Finalmente, se intenta integrar un nodo implementado mediante el cliente Besu dentro de la arquitectura de nodos de la Red T de Alastria. Para conseguir esto,



se deben tener en cuenta todos los elementos considerados anteriormente y que son imprescindibles para que dos nodos, implementados mediante diferentes clientes, se puedan conectar entre ellos.

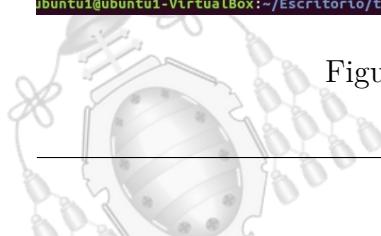
En este caso, lo primero que se debe tener en cuenta es que la Red T de prueba implementada anteriormente no utiliza un archivo génesis para definir la configuración de la red, si no que utiliza la configuración por defecto que tiene la red. Entonces, este es el primer cambio que se debe considerar dentro de la implementación de la Red T, ya que ahora es necesario definir un archivo génesis para que sea igual que el que se utilice con el nodo implementado mediante Besu. El principal problema que supone tener que introducir un archivo génesis en la configuración de Alastria es que es necesario modificar los scripts ya programados y la estructura de directorios planteada por Alastria.

Para recordar la estructura de directorios utilizada para la implementación de la Red T, se debe volver a la figura 4.1 en la que se muestra un esquema de todos los directorios y archivos más importantes que forman parte de la Red T. En este caso, primero se debe trabajar sobre el directorio network. En este directorio es donde se almacena la información de la cadena de bloques y, por tanto, el origen de la cadena. En este caso, se debe iniciar la cadena de bloques utilizando el archivo génesis adecuado para que la cadena de bloques tenga la configuración correcta.

En la figura 4.46 se muestra un ejemplo de realización del inicio de la cadena. En este caso se ha realizado de manera manual, pero realmente esto se debe integrar dentro del script *start_node.sh* para que se pueda ejecutar de manera automática cada vez que se inicie la red. La integración de este proceso dentro del script se muestra en la figura 4.47.

```
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/test-environment-master/infrastructure/testnet/network$ sudo geth --datadir main/ init ..genesis.json
Comando utilizado para la iniciación
WARN [06-28|16:55:25] No etherbase set and no accounts found as default
INFO [06-28|16:55:25] Allocated cache and file handles          database=/home/ubuntu1/Escritorio/test-environment-master/infrastructure/testne
t/network/main/geth/chaindata cache=16 handles=16
INFO [06-28|16:55:25] Writing custom genesis block
INFO [06-28|16:55:25] Successfully wrote genesis state           database=chaindata
hash=e1278d...16e4a1 Hash del nuevo bloque cero creado
INFO [06-28|16:55:25] Allocated cache and file handles          database=/home/ubuntu1/Escritorio/test-environment-master/infrastructure/testne
t/network/main/geth/lightchaindata cache=16 handles=16
INFO [06-28|16:55:25] Writing custom genesis block
INFO [06-28|16:55:25] Successfully wrote genesis state           database=lightchaindata
hash=e1278d...16e4a1
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/test-environment-master/infrastructure/testnet/network$
```

Figura 4.46.- Inicio de la cadena de bloques





```
RPC_OPTIONS="--rpc --rpccaddr 0.0.0.0 --rpccapi admin,cli,eth,debug,miner,net,txpool,personal,web3,quorum,istanbul --rpccport 2200$PUERTO"
GENERAL="--identity $IDENTITY --networkid 160616 --ethstats $IDENTITY:bb98a0b6442386d0cdf8a31b267892c1@$ETH_STATS_IP:3000 "
NETWORK="--port 2100$PUERTO"
MINING="--mine --minerthreads 1 --syncmode "full""
DATA_DIR="--datadir "${PWD}"/network/"$NODE_NAME""
EXTRA="--miner.gaslimit 00000000"

if [ "$NODE_NAME" == "main" -o "$NODE_NAME" == "validator1" -o "$NODE_NAME" == "validator2" ]; then
    rm -rf "${PWD}"/network/"$NODE_NAME"
    mkdir "${PWD}"/network/"$NODE_NAME"
    cp permissioned-nodes.json "${PWD}"/network/"$NODE_NAME/permissioned-nodes.json"
    cp static-nodes.json "${PWD}"/network/"$NODE_NAME/static-nodes.json"
    geth ${DATA_DIR} init genesis.json Esta es la línea que permite el nuevo inicio de la cadena de bloques
    nohup geth ${DATA_DIR} $GENERAL $NETWORK $RPC_OPTIONS 2 >> "${PWD}"/logs/quorum_"$NODE_NAME"_"${_TIME}" .log &
```

Esta parte que se añade al script original está pensada para gestionar el borrado de posibles orígenes de cadena anteriores y también gestionar los ficheros de permisionado utilizados.

Figura 4.47.- Script start_node.sh modificado

Comparando lo que se observa en la figura 4.47 con lo mostrado en las figuras 4.7 y 4.8 se puede apreciar que se ha añadido alguna línea al script. Estas líneas permiten gestionar de manera adecuada los directorios de la Red T de Alastria, así como los archivos de permisionado adecuados. Se debe borrar antes de cada inicio la información almacenada previamente en el directorio designado para la cadena de bloques, ya que de lo contrario, es posible que se produzcan incompatibilidades debido a diferentes bloques origen. En la figura 4.48 se muestra un ejemplo los fallos surgidos a la hora de definir un nuevo bloque génesis porque ya existe otro que ha sido registrado previamente, aspecto que ha surgido ocasionalmente a lo largo del proyecto.

```
ubuntu1@ubuntu1-VirtualBox:~/Escritorio/test-environment-master/infrastructure/testnet/logs$ cat quorum_main_20200624095402.log
WARN [06-24|09:54:03] No etherbase set and no accounts found as default
INFO [06-24|09:54:03] Allocated cache and file handles database=/home/ubuntu1/Escritorio/test-environment-master/infrastructure/testnet/network/main/geth/chaindata cache=16 handles=16
Fatal: Failed to write genesis block: database already contains an incompatible genesis block (have 77a8c93fdee76dcb, new 255bbd72f45ae09e)
```

Figura 4.48.- Error con el inicio de la cadena

Siguiendo con lo visto en la figura 4.47, se debe destacar la línea que permite iniciar la cadena de bloques con el archivo génesis correcto. Esta línea se ejecuta una vez que se han borrado todos los archivos relacionados con otros inicios de la cadena de bloques definidos previamente. También se debe recalcar que esta configuración es adecuada para las pruebas que se realizan en este proyecto. En caso de pretender realizar esto para entornos más reales, seguramente la configuración y la forma de programar esto debería ser completamente diferente.

Desde la parte del nodo implementado mediante el cliente Besu, la configuración es similar a la realiza en apartados anteriores, ya que el nodo Besu simplemente se conecta a los nodos de la Red T. En la figura 4.49 se muestra el inicio del nodo de



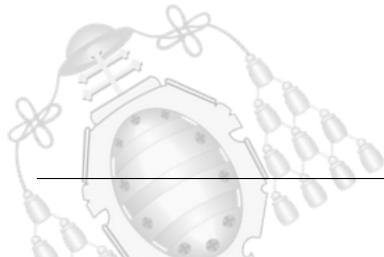
Besu. En esta figura, se remarcán dos partes principales. La primera es el comando ejecutado para invocar el cliente Besu. Recibe como parámetro el enode de uno de los nodos implementados con Geth. Pero, esto es opcional, ya que funciona sin necesidad de esto. Y la segunda parte que se remarcó es el enode del nodo implementado con Besu que se está ejecutando. Principalmente se ha remarcado para que sirva como prueba de que el funcionamiento de la comunicación entre los distintos nodos es correcta y que además permita verificar la coherencia en todo el desarrollo planteado.

```
buntu1@ubuntu1-VirtualBox:~/Escritorio/IBFT-network$ ./ConexionREDT.sh "enode://0105a6f7a3bb06b17732fe7ef687e7f2ce4f4dd8ae1ad88e16f0cdcc0a66b165b6ae382f08bcba91c9c510361572ceeb890e8b74d59728c0fad6584373414092.185.221.150:21000"
2020-06-29 11:23:03.199+02:00 | main | INFO | Besu | Starting Besu version: besu/v1.4.1/linux-x86_64/openjdk-java-11
2020-06-29 11:23:03.512+02:00 | main | INFO | StaticNodesParser | StaticNodes file /home/ubuntu1/Escritorio/IBFT-network/Node2/data/static-nodes.json does not exist, no static connections will be created.
2020-06-29 11:23:03.512+02:00 | main | INFO | Besu | Connecting to 0 static nodes.
2020-06-29 11:23:04.584+02:00 | main | INFO | KeyPairUtil | Loaded key 0x10337904bf1b57bf161f0fa46fa3609cfec4621c8b4178443ced8618260b6f656c493646b9a018d6c5cc2dc042a7b7d74fe07a6b49c43b258f98de23dfed7aa from /home/ubuntu1/Escritorio/IBFT-network/Node2/data/key
2020-06-29 11:23:04.629+02:00 | main | INFO | DatabaseMetadata | Lookup database metadata file in data directory: /home/ubuntu1/Escritorio/IBFT-network/Node2/data
2020-06-29 11:23:04.718+02:00 | main | INFO | RocksDBKeyValueStorageFactory | Existing database detected at /home/ubuntu1/Escritorio/IBFT-network/Node2/data. Version 1
2020-06-29 11:23:05.074+02:00 | main | INFO | ProtocolScheduleBuilder | Protocol schedule created with milestones: [ConstantinopleFix: 0]
2020-06-29 11:23:05.281+02:00 | main | INFO | RunnerBuilder | Detecting NAT service.
2020-06-29 11:23:05.948+02:00 | main | INFO | Runner | Starting Ethereum main loop ...
2020-06-29 11:23:05.948+02:00 | main | INFO | NatService | No NAT environment detected so no service could be started
2020-06-29 11:23:05.949+02:00 | main | INFO | NetworkRunner | Starting Network.
2020-06-29 11:23:06.086+02:00 | nioEventLoopGroup-2-1 | INFO | RLpxAgent | P2P RLpx agent started and listening on /0:0:0:0:0:0:0:30304.
2020-06-29 11:23:06.086+02:00 | main | INFO | PeerDiscoveryAgent | Starting peer discovery agent on host=0.0.0.0, port=30304
2020-06-29 11:23:06.188+02:00 | vert.x-eventloop-thread-1 | INFO | VertxPeerDiscoveryAgent | Started peer discovery agent successfully, on effective host=0.0.0.0:0:0:0:0:0 and port=30304
2020-06-29 11:23:06.181+02:00 | main | INFO | PeerDiscoveryAgent | P2P peer discovery agent started and listening on /0:0:0:0:0:0:0:30304
2020-06-29 11:23:06.255+02:00 | main | INFO | DefaultP2PNetwork | Enode URL enode://10337904bf1b57bf161f0fa46fa3609cfec4621c8b4178443ced8618260b6f656c493646b9a018d6c5cc2dc042a7b774fe07a6b49c43b258f98de23dfed7aa@192.168.1.35:30304
2020-06-29 11:23:06.269+02:00 | main | INFO | DefaultSyncronizer | Starting synchronizer.
2020-06-29 11:23:06.270+02:00 | main | INFO | FullSyncDownloader | Start full sync.
```

Figura 4.49.- Inicio del nodo de Besu

Finalmente, se han iniciado dos nodos pertenecientes a la Red T de prueba y el nodo implementado mediante Besu. A continuación, se muestran una serie de figuras que permiten demostrar que se ha conseguido realizar la conexión entre uno de los nodos implementados mediante Geth con el nodo implementado mediante Besu. Se ha realizado así, en lugar de realizar la conexión de todos con todos, porque es más parecido a una situación real, ya que en la Red T original, es normal que cada nodo tengan un número de nodos diferentes conectados a él.

En la primera figura de esta lista, figura 4.50, se muestra la imagen de las estadísticas de la Red T. En este caso, lo que se remarcó es el número de nodos que tiene cada uno de los dos nodos que pertenecen a la Red T. Se ve como el nodo Main tiene dos nodos conectados, mientras que el validator1 solo tiene un nodo conectado.



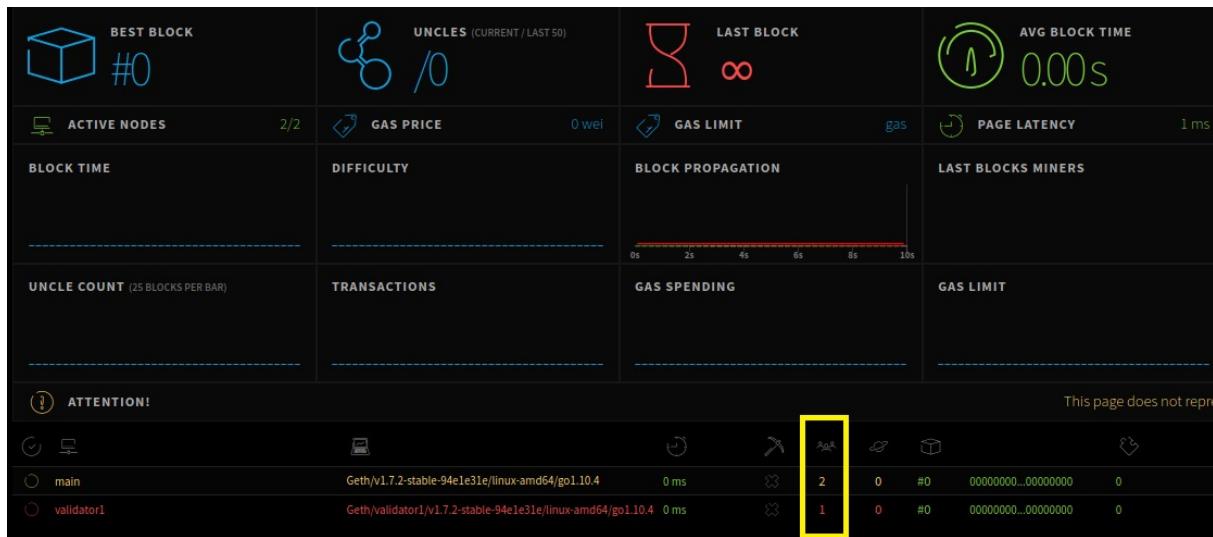
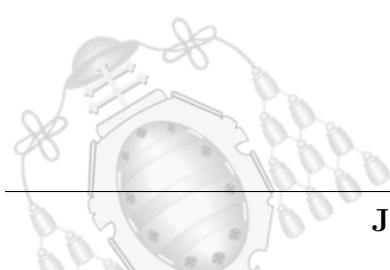


Figura 4.50.- Gráfico de los nodos conectados

En la figura 4.51 se muestra el resultado de usar la consola interactiva de Javascript del nodo Main. Se ve como el nodo Main tiene dos nodos conectados directamente. Uno de ellos es un nodo implementado con Geth (validator1) y otro de ellos es el nodo implementado con Besu. Se observa como el enode del nodo de Besu coincide con el enode mostrado en la figura 4.49. Otro elemento que es destacable de esta imagen es la coincidencia del campo *genesis* para ambos nodos conectados al nodo Main. Este campo representa el Hash del bloque cero o bloque génesis. En caso de que este campo fuese diferente en alguno de los casos, no se podría realizar la conexión, ya que es estrictamente necesario que dos nodos tengan el mismo origen de la cadena de bloques para poder conectarse entre ellos.





```
> admin.peers
[{
  caps: ["IBF/1", "eth/62", "eth/63", "eth/64"],
  id: "18337904bf1fb57bf161f0fa6fa3a509cfe4621c8b4178443ced8618260b6f656c493646b9a018d6c5cc2dc042a7b7747fe07a6b49c43b258f98de23dfedb7aa",
  name: "besu/v1.4.1/linux-x86_64/openjdk-java-11",
  network: {
    localAddress: "192.168.1.83:21000",
    remoteAddress: "92.185.221.150:32944"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xe1278dc7798427ddebfbe39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1", Hash del bloque cero que utiliza este nodo
      version: 03
    }
  }
}, {
  caps: ["eth/63"],
  id: "911ed40021540d69b4a2199e8ea942c39d888d2f21db387475a2adf6cf2eae2407806f6e94b36727cd885bde19c1f964d135a3d530809d95da16eb83ca44e44",
  name: "Geth/validator1/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:35500",
    remoteAddress: "92.185.221.150:21005"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xe1278dc7798427ddebfbe39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1", Hash del bloque cero que utiliza este nodo
      version: 03
    }
  }
}]
```

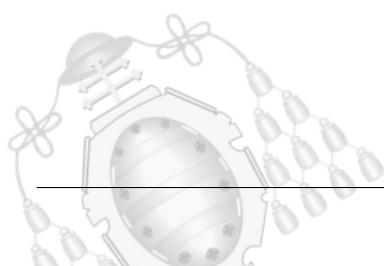
Figura 4.51.- Información del nodo Main

Por otro lado, en la figura 4.52 se muestra la misma información de los nodos conectados al nodo validator1. En este caso, solo presenta información sobre el nodo Main. Esto se debe a que solo tiene un nodo conectado, ya que el permisionado para el nodo implementado con Besu no se ha activado para este nodo.

```
> admin.peers
[{
  caps: ["eth/63"],
  id: "0105a6f7a3bb6b17732fe7ef687e7f2ce4f4dd8ae1ad88e16f@cdcc0a66b165b6ae382f08bcba91c9c510361572ceebb8900e8b74d59728c0fad65843734314",
  name: "Geth/main/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
  network: {
    localAddress: "192.168.1.83:21005",
    remoteAddress: "92.185.221.150:35500"
  },
  protocols: {
    eth: {
      difficulty: 16,
      head: "0xe1278dc7798427ddebfbe39f2395f9c2f09d8fc5a79f89fcbe232c1c5616e4a1", Hash del bloque cero que utiliza
      version: 03
    }
  }
}]
```

Figura 4.52.- Información del nodo valitador1

En la figura 4.53, se muestra la misma información que en los casos anteriores, pero desde el punto de vista del nodo implementado mediante Besu. Para obtener la información de este nodo se utiliza la API de JSON-RPC-HTTP.

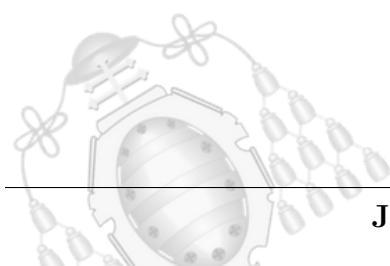




```
ubuntu1@ubuntu1-VirtualBox:~$ curl -X POST --data '{"jsonrpc":"2.0","method":admin_peers,"params":[],"id":1}' http://127.0.0.1:8546
{
    "jsonrpc" : "2.0",
    "id" : 1,
    "result" : [ [
        "version" : "0x5",
        "name" : "Geth/main/v1.7.2-stable-94e1e31e/linux-amd64/go1.10.4",
        "caps" : [ "eth/63" ],
        "network" : {
            "localAddress" : "192.168.1.35:32944",
            "remoteAddress" : "92.185.221.150:21000"
        },
        "port" : "0x0",
        "id" : "0x0105a6f7a3bb06b17732fe7ef687e7f2ce4f4dd8ae1ad88e16f0cdcc0a66b165b6ae382f08bcba91c9c510361572ceebb8900e8b74d59728c0fad65843734314
    ] ]
}
```

Figura 4.53.- Información del nodo implementado con Besu

Finalmente, después de haber realizado la implementación de los tres nodos y comprobar que el funcionamiento sigue el comportamiento esperado, se puede garantizar que es posible conectar los nodos de la Red T con un nodo implementado mediante el cliente Besu. Esta investigación realizada, sirve como puerta de entrada para investigaciones posteriores en las que se ponga esta red homogénea en funcionamiento y se compruebe su funcionamiento en diferentes situaciones.



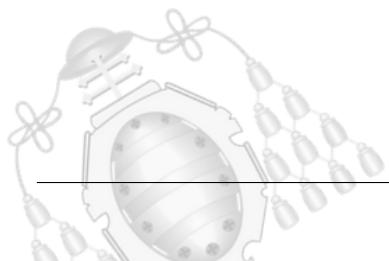


5. Planificación

En este capítulo se muestra la planificación utilizada para la realización de este proyecto. Al principio, se realiza una división del mismo en tres fases diferentes.

1. La primera fase consiste en una fase familiarización con la tecnología de blockchain y la asociación de Alastria. Empezando por los conceptos más simples y profundizando en todos los elementos más importantes para el desarrollo del proyecto.
2. La segunda fase consiste en la implementación de una Red T de prueba y una Red B. De esta forma se pretende entender qué parámetros son los esenciales para controlar correctamente el funcionamiento de ambas redes y cuáles tienen mayor relevancia a la hora de buscar la interoperabilidad de clientes. Además, realizar la implementación de estas redes de prueba también sirve como forma de poner en práctica todos los conocimientos aprendidos en la fase 1.
3. La tercera fase del proyecto consiste en buscar la interoperabilidad entre los dos tipos de clientes a partir de toda la información recogida anteriormente. Gracias a la información conseguida en las fases uno y dos se pretenden conseguir la interoperabilidad entre los clientes Geth y Besu.

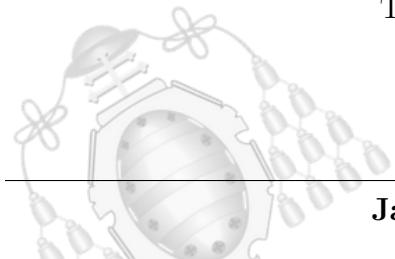
Dentro de estas tres fases, se realizan una serie de tareas que permiten dividir toda la carga de trabajo del proyecto en metas temporales. Para reflejar la planificación realizada de estas tareas a lo largo de la duración del proyecto se añade el diagrama de Gantt mostrado en la figura 5.1. También se incorpora la tabla 5.1 en la que se muestra la lista de todas las tareas realizadas y cómo se han organizado mediante la elaboración de la estructura desagregada de trabajos (EDT).





EDT	Nombre de la Tarea
1	Elección temática del proyecto
2	Estudio de los conceptos básicos de blockchain
2.1	¿Qué es blockchain?
2.2	Tipos de nodos en la red
2.3	Principios básicos blockchain
2.4	Tipos de algoritmo de consenso
2.5	Funcionamiento de la cadena de bloques
2.5.1	Función Hash y árbol de Merkle
2.5.2	Generación de nuevos bloques
3	Estudio de tipos de redes
3.1	Estudio de las redes públicas
3.2	Estudio de las redes privadas
3.3	Estudio redes permisionadas
4	Estudio Red Ethereum
4.1	Principios básicos de Ethereum
4.2	Aplicación Red Quorum
5	Estudio Red Quorum
5.1	Análisis características básicas de Quorum
5.2	Aplicación Red Quorum
6	Análisis de Alastria
6.1	Características sociales y económicas de Alastria
6.2	Características técnicas de Alastria
7	Preparación entorno de desarrollo
7.1	Montaje Red T
7.2	Montaje Red B
8	Investigación de sincronización entre Red T y Red B
9	Redacción memoria del proyecto

Tabla 5.1.- Tareas del proyecto



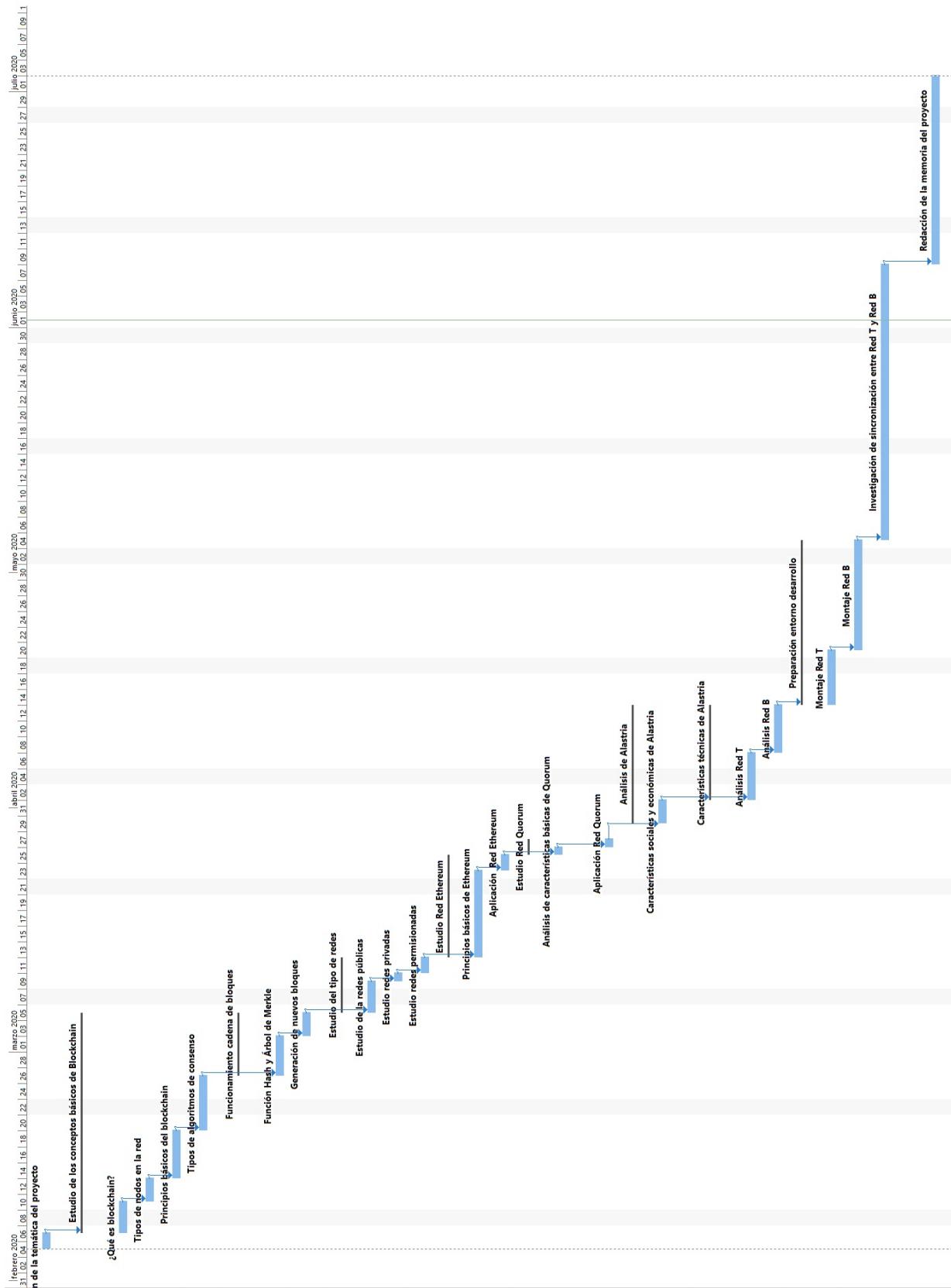
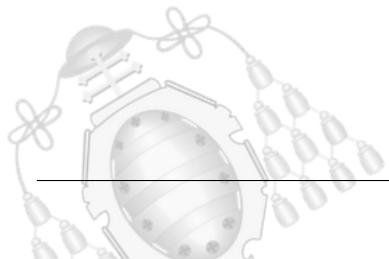


Figura 5.1.- Diagrama de Gantt



Javier Rodríguez Fernández



6. Conclusiones

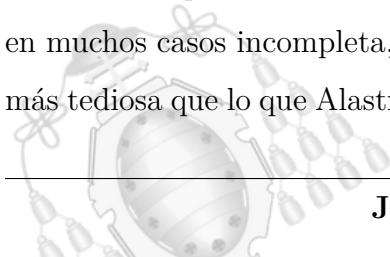
En este capítulo se presentan las conclusiones del proyecto desde dos puntos de vista. Primero, desde un punto de vista técnico y más relacionado con el objetivo principal del proyecto. Y segundo, desde un punto de vista personal, relacionado con todos los conocimientos aprendidos y todo la práctica obtenida con esta tecnología.

6.1.- Conclusiones técnicas

La primera conclusión técnica que se puede sacar es que el concepto de blockchain es un concepto muy amplio que abarca una enorme cantidad de elementos, y que por tanto es muy difícil ser experto en la tecnología de blockchain en general. Dentro de ella se pueden estudiar conceptos tan diferentes como por ejemplo: los diferentes tipos de redes que existen según el propósito y según la privacidad la red, los distintos algoritmos de consenso que se utilizan, los diferentes clientes existentes para implementar los nodos, los smarts contracts y las aplicaciones descentralizadas (Dapps).

Por otro lado, y ya enfocándose más en el objetivo del proyecto se puede destacar que el consorcio de Alastria presenta una opción interesante para la integración de la tecnología de blockchain en un ámbito multisectorial. Esto permite que esta tecnología gane relevancia en la evolución de muchos servicios. Además, con el planteamiento de buscar una única red homogénea y que sea compatible con diferentes tipos de clientes permite que esta red pueda llegar a muchos más clientes y que se eviten muchas restricciones ligadas al uso de una única tecnología en concreto.

Desde el punto de vista de implementación de las redes, se puede concluir que la Red T tiene una estructura bastante cerrada, en la que es complejo modificar algo a partir de la información que proporciona Alastria, ya que tiene un árbol de directorios bastante compacto. Además, que la información proporcionada por Alastria es escasa y en muchos casos incompleta, de forma que la implementación de esta Red T es mucho más tediosa que lo que Alastria presenta. En cambio, el caso de la implementación de la





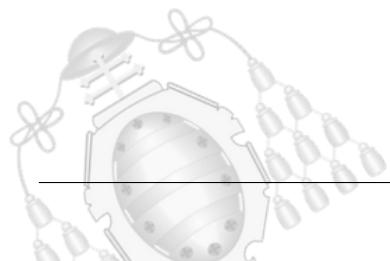
Red B es bastante más flexible y la información proporcionada por Hyperledger Besu en su página web oficial facilita considerablemente la implementación de esta Red.

Por otra parte, en relación a buscar la interoperabilidad entre clientes, se puede concluir que es algo viable, ya que al principio del proyecto no se sabía si ese objetivo podría ser conseguido. Se ha demostrado en este proyecto que teniendo en cuenta los parámetros adecuados, dos nodos implementados mediante clientes Geth y Besu pueden ser compatibles. Esto es un punto interesante ya que no existe ningún estudio ni análisis previo a tal efecto.

6.2.- Conclusiones personales

Desde un punto de vista personal, las conclusiones obtenidas de este proyecto son todas positivas. Primero, se ha aprendido mucho sobre la tecnología del blockchain en general. Desde los conocimientos más básicos hasta conceptos mucho más complejos y que son parte de esencial del funcionamiento de la tecnología de blockchain.

Por otro lado, se ha desarrollado todo el proyecto, partiendo de cero y realizando todo el proceso de forma que se fuese entendiendo todo lo que se realizaba. Esto permite obtener un elevado grado de conocimiento sobre todo lo que se ha realizado. Además, ha servido para poner en práctica habilidades como el auto-aprendizaje o la capacidad de síntesis de información.





7. Líneas futuras

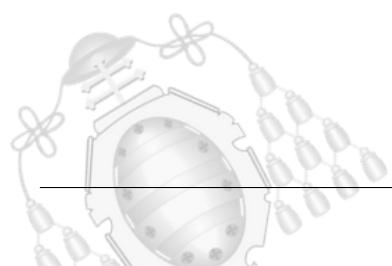
Según se ha comentado en capítulos anteriores, la tecnología de blockchain es una tecnología muy amplia y que tiene muchas áreas de investigación y desarrollo posibles. Además, al tratarse de una tecnología que aún no ha llegado a su máximo auge permite que queden muchas vías de investigación aún sin explorar en profundidad. Además, con la previsión de que esta tecnología sea fundamental en el futuro para el desarrollo de muchos servicios, aumenta el interés y los posibles beneficios a la hora de realizar investigaciones sobre esta tecnología y alguno de sus diversos campos.

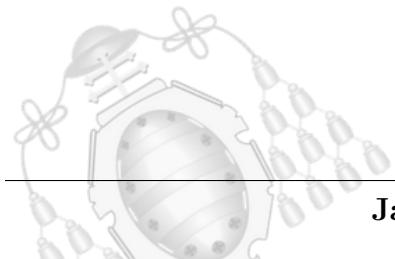
A continuación se plantea una lista de posibles proyectos, pensados para ser desarrollados en entornos simulados, a partir de lo estudiado en este:

- Búsqueda de la interoperabilidad entre clientes Geth y Besu, pero en lugar de utilizar un bootnode implementado mediante el cliente Geth, implementar dicho bootnode mediante el cliente Besu. Esto tendría total aplicación en una Red B en la que se pretenda integrar algún nodo implementado mediante el cliente Geth.
- Integración de los clientes Geth y Besu con otro tipo de clientes para ampliar el rango de posibles tecnologías a utilizar y tratar de aplicar eso a la red de Alastria.
- Implementación de una Red T y una Red B de pruebas de Alastria. Análisis de todos los parámetros de configuración de la red. Realización de pruebas de funcionamiento en diferentes situaciones y comparación de los resultados obtenidos en cada red. Todo estos análisis llevarían a obtener una comparativa de la eficiencia de cada tipo de red en diferentes escenarios.
- Desarrollo de una aplicación descentralizada sobre la Red T de prueba de Alastria. Esto requiere un gran dominio de los smart contracts, de conocimientos de la tecnología de blockchain, además de desarrollo de software para realizar el front-end de la aplicación.
- Igual que el caso anterior, pero en lugar de desarrollar la aplicación descentralizada sobre la Red T de prueba de Alastria, realizarla sobre la Red B.



Por otro lado, todo este tipo de proyectos, pueden ser planteados para implementarse sobre las redes originales de Alastria o sobre cualquier otra tipo de red de blockchain pública existente en la actualidad. Pero en caso de realizarse de esta forma, requieren mucha más precisión a la hora de la configuración de todos los elementos, así como mayor capacidad computacional y de almacenamiento en el sistema en el que se desarrollen.

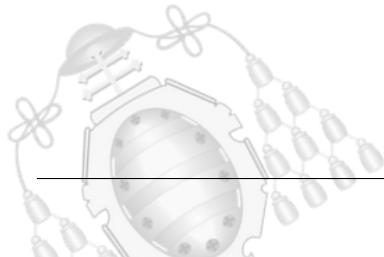






Bibliografía

- [1] Origen blockchain, <https://www.binance.vision/es/blockchain/history-of-blockchain> Visitado 12/05/20
- [2] Información sobre Árbol de Merkle, <https://academy.bit2me.com/que-es-un-arbol-merkle/> Visitado 13/05/20
- [3] Información sobre principios del blockchain, <https://101blockchains.com/introduction-to-blockchain-features/> Visitado 13/05/20
- [4] Información sobre algoritmos de consenso, <https://www.geeksforgeeks.org/consensus-algorithms-in-blockchain/> Visitado 13/05/20
- [5] Información sobre el uso de HASH en blockchain, <https://academy.bit2me.com/que-es-hash/> Visitado 16/05/20
- [6] Información sobre el algoritmo de consenso proof of work, <https://es.cointelegraph.com/explained/proof-of-work-explained> Visitado 17/05/20
- [7] Información sobre la función criptográfica Hash en blockchain, <https://www.criptonoticias.com/cryptopedia/blockchain-bloques-transacciones-firmas-digitales-hashes/> Visitado 17/05/20
- [8] Información sobre el ataque de 51 % en blockchain, <https://www.criptonoticias.com/cryptopedia/blockchain-bloques-transacciones-firmas-digitales-hashes/> Visitado 17/05/20
- [9] Información sobre el algoritmo de consenso Proof of Authority, <https://academy.bit2me.com/que-es-proof-of-authority-poa/> Visitado 17/05/20

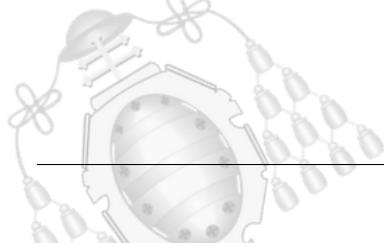




- [10] Información sobre el algoritmo de consenso Proof of Authority, <https://www.binance.vision/es/blockchain/proof-of-authority-explained> Visitado 18/05/20
- [11] Información sobre el algoritmo de consenso Proof of Stake, <https://academy.bit2me.com/que-es-proof-of-stake-pos/> Visitado 18/05/20
- [12] Información sobre la seguridad del algoritmo de consenso Proof of Stake, <https://www.investopedia.com/terms/p/proof-stake-pos.asp> Visitado 18/05/20
- [13] Información sobre el algoritmo de consenso IBFT, [http://docs.goquorum.com/en/latest/Consensus\(ibft\)/ibft/](http://docs.goquorum.com/en/latest/Consensus(ibft)/ibft/) Visitado 10/06/20
- [14] Información sobre la dificultad en blockchain, <https://es.cointelegraph.com/explained/what-is-the-difficulty-of-mining> Visitado 20/05/20
- [15] Información sobre las recompensas en blockchain, <https://es.cointelegraph.com/explained/what-is-the-block-reward> Visitado 20/05/20
- [16] Información sobre los contratos inteligentes, <https://academy.bit2me.com/que-son-los-smart-contracts/> Visitado 21/05/20
- [17] Información sobre la seguridad y los contratos inteligentes, <https://ayudaleyprotecciondatos.es/2019/07/25/smart-contracts-ejemplos/> Visitado 21/05/20
- [18] Información sobre las aplicaciones descentralizadas, <https://academy.bit2me.com/que-son-las-dapps/> Visitado 21/05/20
- [19] Información sobre los tipos de redes de blockchain, <https://www.alisys.net/es/blog/blockchain-tipos-de-redes-y-aplicaciones-de-uso> Visitado 22/05/20
- [20] Información sobre las redes permisionadas de blockchain, <http://abancainnova.com/es/opinion/los-tipos-de-blockchain-publica-privada-o-consorcio-explicados/> Visitado 23/05/20

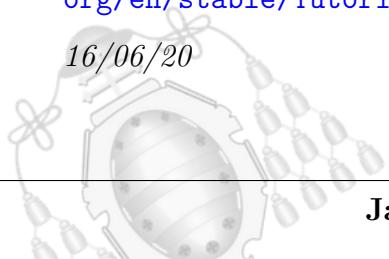


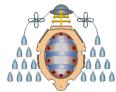
- [21] Información sobre Fintech y blockchain, <https://es.cointelegraph.com/news/fintech-means-blockchain-is-there-anything-else> Visitado 22/05/20
- [22] Información sobre el origen de Ethereum, <https://es.bitcoinwiki.org/wiki/Ethereum> Visitado 23/05/20
- [23] Información sobre características de Ethereum, <https://ethereum.org/what-is-ethereum/> Visitado 23/05/20
- [24] Información sobre la criptomonedra de Ethereum <https://es.cointelegraph.com/ethereum-for-beginners/what-is-ethereum> Visitado 23/05/20
- [25] Información sobre el gas en Ethereum, <https://www.miethereum.com/ether/gas/> Visitado 24/05/20
- [26] Información sobre Quorum, <http://docs.goquorum.com/en/latest/> Visitado 24/05/20
- [27] Información sobre el permisionado de Quorum, <http://docs.goquorum.com/en/latest/Permissioning/> Visitado 24/05/20
- [28] Información sobre el algoritmo de consenso en Quorum, <https://medium.com/everis-blockchain> Visitado 24/05/20
- [29] Información sobre el algoritmo de consenso Raft, <http://docs.goquorum.com/en/latest/Consensus/raft/raft/> Visitado 25/05/20
- [30] Información de comparativa entre Ethereum y Quorum, <https://comunytek.com/comparativa-tecnologias-blockchain/> Visitado 24/05/20
- [31] Información sobre Alastria, <https://alastria.io/> Visitado 27/05/20
- [32] Información sobre Alastria y su origen, <https://www.grantthornton.es/perspectivas/blockchain/nace-red-lyra-la-red-blockchain-espanola-multisectorial-de-referencia-en-el-mundo> Visitado 27/05/20





- [33] Presentación general de Alastria, https://alastria.io/wp-content/uploads/2020/05/20200520_Alastria-Corporate-presentation_ESP.pdf Visitado 27/05/20
- [34] Información sobre la red de Alastria, <https://alastria.io/la-red/> Visitado 28/05/20
- [35] Información sobre los socios de la Red de Alastria, <https://alastria.io/socios-nodos/> Visitado 28/05/20
- [36] Información sobre la Red B de Alastria, <https://www.blockchaineconomia.es/red-besu-de-alastria/> Visitado 28/05/20
- [37] Información sobre instalación de red T, <https://github.com/alastria/test-environment> Visitado 02/06/20
- [38] Información sobre los clientes de la red T, https://medium.com/@alastria_es/comparativa-de-plataformas-dlt-2fc425c30204 Visitado 03/06/20
- [39] Información sobre línea de comandos del cliente Geth, <https://github.com/ethereum/go-ethereum/wiki/Command-Line-Options> Visitado 11/06/20
- [40] Información sobre cliente la consola JavaScript de Geth, <https://github.com/ethereum/go-ethereum/wiki/JavaScript-Console> Visitado 12/06/20
- [41] Información sobre las APIs JSON-RPC que permite Geth, <https://geth.ethereum.org/docs/rpc/server> Visitado 12/06/20
- [42] Información sobre la API istabul del cliente Geth, <https://github.com/getamis/go-ethereum/wiki/RPC-API> Visitado 14/06/20
- [43] Información sobre la votación en la API istanbul, <https://medium.com/getamis/istanbul-bft-ibft-c2758b7fe6ff> Visitado 14/06/20
- [44] Información sobre la implementación de la Red B, <https://besu.hyperledger.org/en/stable/Tutorials/Private-Network/Create-IBFT-Network/> Visitado 16/06/20





- [45] Información sobre la instalación de Besu, <https://besu.hyperledger.org/en/stable/HowTo/Get-Started/Install-Binaries/> Visitado 16/06/20
- [46] Información sobre las características básicas de Hyperledger Besu, <https://besu.hyperledger.org/en/stable/> Visitado 16/06/20
- [47] Información sobre las principales fortalezas de Hyperledger Besu, <https://pegasys.tech/solutions/hyperledger-besu> Visitado 16/06/20
- [48] Información sobre la línea de comandos de Hyperledger Besu, <https://besu.hyperledger.org/en/stable/Reference/CLI/CLI-Syntax/> Visitado 16/06/20
- [49] Información sobre la API JSON-RPC de Hyperledger Besu, <https://besu.hyperledger.org/en/stable/Reference/API-Methods/> Visitado 16/06/20
- [50] Información sobre el archivo genesis, <https://medium.com/taipei-ethereum-meetup/beginners-guide-to-ethereum-3-explain-the-genesis-file-> Visitado 20/06/20
- [51] Información sobre los parámetros del archivo genesis, <https://www.asynclabs.co/blog/params-in-ethereum-genesis-block-explained/> Visitado 20/06/20
- [52] Información sobre Metamask, <https://academy.bit2me.com/que-es-metamask-la-forma-mas-facil-de-usar-dapps/> Visitado 21/06/20
- [53] Información sobre puesta en marcha de un bootnode, <https://github.com/ethereum/go-ethereum/wiki/Setting-up-private-network-or-local-cluster> Visitado 26/06/20

