Universitat Autònoma
de Barcelona

Plataforma para analizar la propagación de datos en la red Bitcoin

Programa	Posgrado de Seguridad en Redes y Sistemas
Tipo documento	Proyecto de Seguridad en Redes y Sistemas
Título documento	Plataforma para analizar la propagación de datos en la red Bitcoin
Alumna	Cristina Molina Rosillo
Directora de proyecto	Cristina Pérez Solà
Fecha	05/06/2018

Índice de contenidos

Índice de ilustraciones

RESUMEN DEL TRABAJO

El presente trabajo consiste en diseñar y crear una plataforma despliegue una red Bitcoin de pruebas, y que permita el análisis de la propagación de la información.

A tal efecto, se ha integrado un esqueleto existente (btc_testbed) que permite la creación de nodos bitcoin en contendores docker, con herramientas de análisis y presentación de métricas, en concreto Graphana y Grafite.

Para poder realizar un análisis enfocado de los flujos de información, realizamos un estudio del funcionamiento de la red Bitcoin, y de sus elementos clave. Para ello, es importante analizar los diferentes protocolos de intercambio de mensajes, y las tipologías de éstos.

ABSTRACT

The present work consists of designing and creating a platform that deploys a Bitcoin network of tests, and that allows the analysis of the propagation of the information.

To this end, an existing skeleton has been integrated (btc_testbed) that allows the creation and management of bitcoin nodes in docker containers, with analysis tools and presentation of metrics, specifically Graphana and Grafite.

In order to perform a focused analysis of information flows, we conducted a study of the operation of the Bitcoin network, and its key elements. For this, it is important to analyze the different message exchange protocols, and its typologies.

1. Introducción

El presente documento expone el trabajo realizado y los resultados conseguidos en el proyecto "Plataforma para analizar la propagación de datos en la red Bitcoin".

En un primera parte se exponen los objetivos y el plan de trabajo, y después se continúa con una explicación de los elementos más relevantes de la red Bitcoin relacionados con el objeto del proyecto.
Después de un apartado que aborda las métricas e indicadores potenciales, se describe el diseño e implementación de la plataforma de pruebas, para acabar finalmente con las conclusiones.

1.1. Motivación del proyecto

La finalidad del proyecto es el estudio del comportamiento del flujo de información de la red Bitcoin. Para ello, se diseñará e implementará una réplica en laboratorio de una red Bitcoin reducida, denominada de aquí en adelante "Plataforma".

El flujo de información a analizar se corresponde con intercambio de información relacionada con los bitcoins, las transacciones, la formación de bloques, y los clientes conectados.

A su vez, los flujos de información se despliegan en una infraestructura de red que se estructura en protocolos de intercambio de mensajes entre nodos. Estos protocolos tendrán que ser estudiados y conocer sus características y parámetros principales para poder contrastarlos con el análisis de comportamiento en nuestra Plataforma.

1.2. Introducción a la red Bitcoin

Bitcoin [1] es un sistema de intercambio económico, o de valor, basado en tecnología blockchain.
Blockchain, gracias a técnicas avanzadas de criptografía, permite efectuar transacciones peer-to-peer de forma registrada y validada, sin necesidad que se gestionen por una entidad intermediadora, y precisamente, su primera aplicación ha sido la transacción de criptomonedas.

El crecimiento de Bitcoin desde sus inicios ha sido exponencial, tanto en número de nodos como en la revalorización de la moneda.

No obstante, durante este período también se han producido episodios importantes de fluctuaciones desmesuradas de su valor y de incidentes de seguridad.

Algunos de los retos que presenta el futuro de Bitcoin son los siguientes [2][3]:

- La gestión de la seguridad depende en gran parte del propio usuario final
- Existen fuertes barreras de entrada para extender el uso de la criptomoneda a más usuarios, por cierta complejidad que presenta
- La privacidad y el anonimato presentan un doble debate:

    - Los partidarios del anonimato completo, no consideran el sistema lo suficientemente anónimo
    - En contraste, el anonimato actual facilita canalizar operaciones ilegales
- El sistema puede ser objeto de creación de monopolios
- La especulación provoca un exceso de volatilidad en el valor de la moneda
- El sistema tiene que ser más escalable para acoger más demanda, y también tiene que presentar un rendimiento adecuado
- La producción de blockchains comporta elevados consumos de máquina de procesamiento, y de subministro eléctrico
- Surgimiento de otros competidores, incluso procedentes de entidades oficiales como bancos, estados, etc.

1.3. Objetivos del proyecto

Para conseguir la finalidad del proyecto, identificamos los siguientes objetivos:

- Diseñar e implementar una Plataforma que simule de la forma más fiel posible el funcionamiento básico de la red Bitcoin real
- Identificar los patrones teóricos de funcionamiento de la red Bitcoin, en cuanto a los flujos de información a analizar (bitcoins, transferencias y comunicaciones entre clientes)
- Comprobar del comportamiento de los flujos de la información en la Plataforma, en función de variaciones en parámetros clave de los patrones teóricos
- Identificar los parámetros clave finales que caracterizan los diferentes flujos en nuestra Plataforma, y que se pueden extrapolar a la propia red Bitcoin

1.4. Metodología a aplicar

La metodología a seguir se basa en una fase de diseño y análisis lo más exhaustiva posible que abarque todos los aspectos clave, que asegure una implementación eficaz, y que obtenga unos indicadores significativos y concluyentes.

El seguimiento del grado de avance en el calendario de proyecto será fundamental para cumplir con los objetivos marcados en cada fase de entrega (PEC).

A continuación, se describen los ejes principales del método de trabajo a seguir.

1.5. Plan de proyecto

La planificación tiene como hitos principales las fechas de entrega de las diferentes PEC. Asimismo, también tiene que estar orientada a que en cada PEC se entregue la parte de proyecto que corresponda.

En concreto, las fases principales del proyecto son las siguientes:

Entrega	Duración	Nombre de la fase
PEC-1	21/02 – 13/03	Plan de trabajo
PEC-2	14/03 – 10/04	Análisis y diseño

PEC-3	11/04 – 08/05	Implementación y pruebas
PEC-4	09/05 – 05/06	Conclusiones y memoria

3.3.1 Principales actividades

Las principales actividades a desarrollar en el proyecto son las siguientes:

1. Diseñar una Plataforma que simule de la forma más fiel posible el funcionamiento básico de la red Bitcoin real.

   a. Documentar el funcionamiento de una red Bitcoin

   Para poder llevar a cabo el diseño hace falta el estudio del funcionamiento actual de la red Bitcoin consultando diferentes fuentes de información.
   Para cada tipo de flujo de información, el interés se centra en los protocolos de comunicaciones, los diferentes mensajes intercambiados, llegando a conocer, por ejemplo:

   - Origen – destino
   - Tipos de mensajes
   - Mecanismos validación mensajes
   - Mecanismos de recuperación ante fallos
   - Mecanismos de criptogafía/cifrado utilizados

   También incluirá investigación sobre experiencias similares a este proyecto en cuanto a simular una red Bitcoin y analizar su comportamiento.

   b. Identificar los patrones teóricos de funcionamiento de la red Bitcoin, en cuanto a los flujos de información a analizar (bitcoins, transferencias y comunicaciones entre clientes)

   Para poder analizar de forma sistemática el comportamiento de la Plataforma, es clave identificar: cada proceso de datos, los mecanismos necesarios para su funcionamiento, y los parámetros clave.

   c. Especificar las pruebas a realizar en la Plataforma

   En fase de diseño es importante especificar todas las pruebas a realizar, en tres ámbitos:

   - Comprobar la correcta puesta en marcha de la Plataforma
   - Comprobar el funcionamiento estándar del sistema
   - Comprobar el funcionamiento del sistema cuando se alteran parámetros clave

   También es necesario identificar cómo se realizarán las diferentes pruebas y qué indicadores serán aplicables, y cómo estarán disponibles en la implementación de la Plataforma.

   Finalmente, también se concretará cómo evaluar las pruebas respecto a los resultados esperados.

   d. Estudio de viabilidad de la implementación de la Plataforma

   La Plataforma requiere el despliegue de infraestructuras y de aplicaciones para su implementación.

En consecuencia, durante el estudio teórico se evaluará la viabilidad de utilizar los recursos dockers, y la de disponer de las aplicaciones y programas necesarios para su puesta en marcha.
Será necesario identificar todos los recursos requeridos para la implementación.

2. Implementación de la Plataforma

   a. Fase de instalación

   Se configurarán los diferentes nodos de la Plataforma, de acuerdo con el diseño establecido, y los recursos necesarios identificados.

   b. Comprobación puesta en marcha

   Se comprobará el correcto funcionamiento de la Plataforma, previo a la realización de las pruebas de comportamiento del sistema, para que cualquier problema técnico o defecto de configuración no alteren las pruebas posteriores.
   También se verificará que los indicadores necesarios para las pruebas están disponibles.

3. Pruebas de comportamiento

Se aplicará el banco de pruebas definido en el diseño, de acuerdo con los dos ámbitos:

- Comprobar el funcionamiento estándar del sistema
- Comprobar el funcionamiento del sistema cuando se alteran parámetros clave

Con los resultados obtenidos, seguramente se producirá un proceso de realimentación y mejora de las pruebas, que conducirá a definir algunas adicionales a las previstas en diseño.

4. Conclusiones sobre las pruebas realizadas

Será necesario recopilar toda la información obtenida, analizarla, y desarrollar las conclusiones pertinentes.

5. Elaboración de la memoria de proyecto

Se abordará la fase de redacción de la memoria, y del material requerido para la presentación y defensa del proyecto.

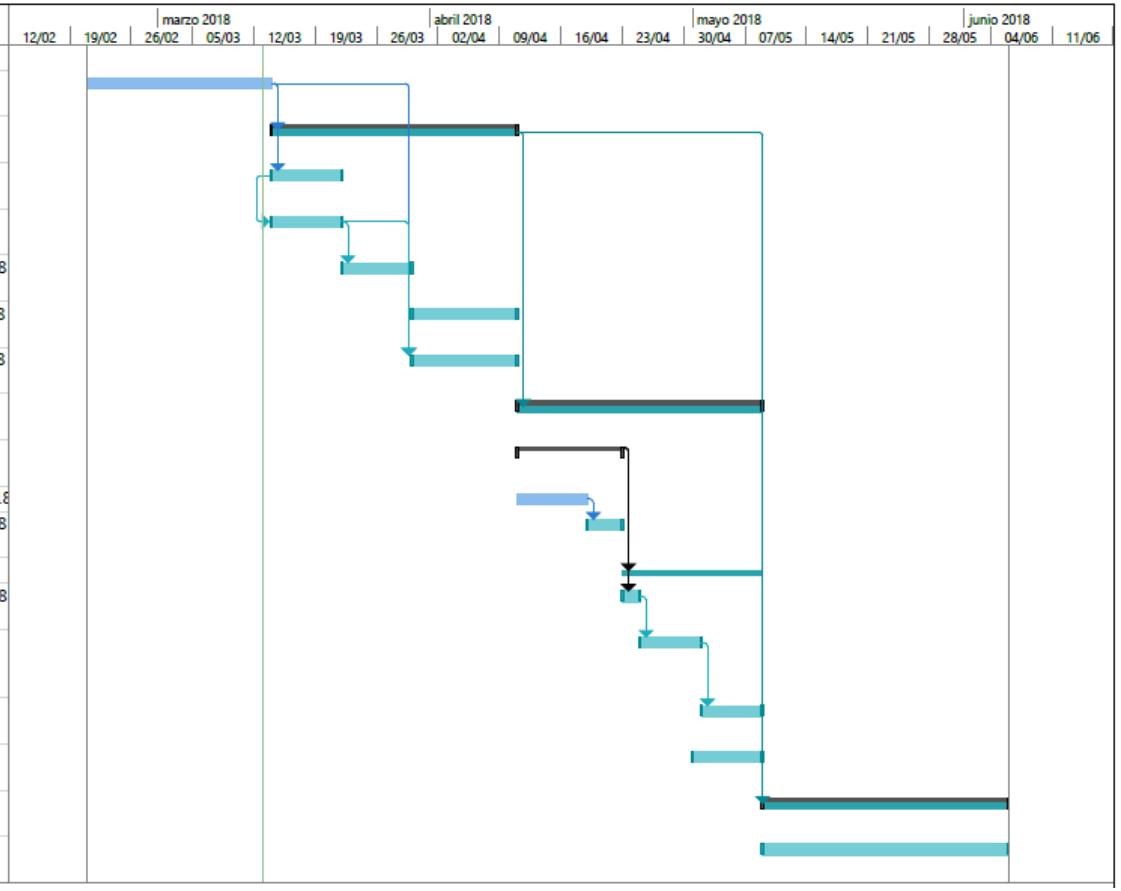
3.3.2 Planificación del proyecto

La relación de todas las tareas con sus duraciones y dependencias es la siguiente:

# Tarea	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Duración	Comentarios
2	**PEC-1. Plan de trabajo**	**15 días**	**mié 21/02/18**	**mar 13/03/18**		**15 días**	
3	**PEC-2. Análisis y diseño**	**20 días**	**mié 14/03/18**	**mar 10/04/18**	**2**	**20 días**	
4	Documentar el funcionamiento de una red Bitcoin	6 días	mié 14/03/18	mié 21/03/18	2	6 días	
5	Identificar los patrones teóricos de funcionamiento de la red Bitcoin	6 días	mié 14/03/18	mié 21/03/18	4CC	6 días	Puede comenzar en paralelo con la tarea 4
6	Especificar las pruebas a realizar en la Plataforma	6 días	jue 22/03/18	jue 29/03/18	5	6 días	Comienza cuando los patrones son claros
7	Estudio de viabilidad de la implementación de la Plataforma	8 días	vie 30/03/18	mar 10/04/18		8 días	
8	Elaboración entregable PEC-2	8 días	vie 30/03/18	mar 10/04/18	2;5	8 días	Puede empezar en paralelo
9	**PEC-3. Implementación y pruebas**	**20 días**	**mié 11/04/18**	**mar 08/05/18**	**3**	**20 días**	
10	**Implementación de la plataforma**	**8 días**	**mié 11/04/18**	**dom 22/04/18**		**8 días**	
11	Fase de instalación	6 días	mié 11/04/18	mié 18/04/18		6 días	
12	Comprobación puesta en marcha	3 días	jue 19/04/18	dom 22/04/18	11	3 días	
13	**Pruebas de comportamiento**				**10**		Comienza con la implementación finalizada
14	Comprobar el funcionamiento estándar del sistema	2 días	lun 23/04/18	mar 24/04/18	10	2 días	
15	Comprobar el funcionamiento del sistema cuando se alteran parámetros clave	5 días	mié 25/04/18	mar 01/05/18	14	5 días	
16	Conclusiones sobre las pruebas realizadas	5 días	mié 02/05/18	mar 08/05/18	15	5 días	
17	Elaboración entregable PEC-2	6 días	mar 01/05/18	mar 08/05/18		6 días	Puede empezar en paralelo
18	**PEC-4. Memoria final del proyecto**	**20 días**	**mié 09/05/18**	**mar 05/06/18**	**3**	**20 días**	
19	Elaboración de la memoria de proyecto	20 días	mié 09/05/18	mar 05/06/18		20 días	

3.3.3 Calendario del proyecto

Id	Modo de tarea	Nombre de tarea	Duración	Comienzo
1				
2		PEC-1. Plan de trabajo	15 días?	mié 21/02/18
3		**PEC-2. Análisis y diseño**	20 días	**mié 14/03/18**
4		Documentar el funcionamiento de una red Bitcoin	6 días	mié 14/03/18
5		Identificar los patrones teóricos de funcionamiento de la red Bitcoin	6 días	mié 14/03/18
6		Especificar las pruebas a realizar en la Plataforma	6 días	jue 22/03/18
7		Estudio de viabilidad de la implementación de la Plataforma	8 días	vie 30/03/18
8		Elaboración entregable PEC-2	8 días	vie 30/03/18
9		**PEC-3. Implementación y pruebas**	20 días	**mié 11/04/18**
10		**Implementación de la plataforma**	8 días	**mié 11/04/18**
11		Fase de instalación	6 días	mié 11/04/18
12		Comprobación puesta en marcha	3 días	jue 19/04/18
13		**Pruebas de comportamiento**		
14		Comprobar el funcionamiento estándar del sistema	2 días	lun 23/04/18
15		Comprobar el funcionamiento del sistema cuando se alteran parámetros clave	5 días	mié 25/04/18
16		Conclusiones sobre las pruebas realizadas	5 días	mié 02/05/18
17		Elaboración entregable PEC-2	6 días	mar 01/05/18
18		**PEC-4. Memoria final del proyecto**	20 días	**mié 09/05/18**
19		Elaboración de la memoria de proyecto	20 días	mié 09/05/18



3.3.4 Plan de riesgos

A continuación, se relacionan los riesgos de proyecto identificados, conjuntamente con las correspondientes medidas mitigadoras:

#ID Riesgo	Fase	Descripción riesgo	Tipo de riesgo	Mitigación
1	PEC-3 Implementación y pruebas	Problemáticas en la implementación de la Plataforma	Alto	- Analizar viabilidad de la Plataforma en fase de diseño
2	PEC-3 Implementación y pruebas	Banco de pruebas no orientado a los objetivos del proyecto	Medio	- Definición de pruebas en fase de diseño de acuerdo con el funcionamiento teórico
3	PEC-3 Implementación y pruebas	Falta de indicadores de seguimiento durante las pruebas	Alto	- Identificación de los indicadores en fase de diseño, y asegurar que en la implementación son trazables en la Plataforma

2. Descripción general de la red Bitcoin

La red bitcoin es una red *peer-to-peer*, en la que cada cliente bitcoin participa conectándose con otros clientes bitcoin. La finalidad de la red bitcoin es propagar transacciones y bloques de *blockchain* a todos los participantes [4].

A grandes rasgos, los elementos clave de una red Bitcoin son los siguientes:

- Usuarios – nodos - conteniendo monederos (*wallets*) con sus claves privadas
- Transacciones que se propagan a través de la red.
- Mineros – nodos - que producen los *blockchain* consensuados, es decir, el registro autoritativo de todas las transacciones.
- La cadena de bloques *blockchain*: en ella se basa todo el sistema de confianza, el registro y autorización de las transacciones.

La característica fundamental de la red Bitcoin es que se basa en que los nodos funcionan de forma equivalente y autónoma, llegando a un consenso gracias a:

- Verificación independiente de cada transacción por cada nodo (tipo *full*)
- Agregación independiente de las transacciones en nuevos bloques, mediante los nodos de minado
- Verificación independiente de los nuevos bloques por cada nodo y ensamblado en una cadena con computación contrastada mediante algoritmo de *proof-of-work*
- Selección independiente, para cada nodo, de la cadena de más procesado computacional, mediante la comprobación del *proof-of-work*

2.1. Los nodos Bitcoin

Cada nodo bitcoin está conectado a la red mediante los nodos que descubre durante el inicio de la sesión *peer-to-peer*, formándose una red mallada, sin jerarquías.

Los nodos pueden tener diferentes roles:

- Función de *routing*: la tienen todos. Consiste en reenviar las transacciones y bloques, siempre que sean válidos
- Base de datos *blockchain*: registro acumulado de todas las transacciones válidas y reconocidas
- *Wallet*: nodo que genera transacciones de bitcoins
- Minero: nodo que crea los bloques de las transacciones que son correctas

Todos los nodos tienen función de *routing* y todos validan y propagan transacciones y bloques, descubriendo y manteniendo conexiones con *peers*.
Se denominan clientes *Bitcoin Core* a los nodos *full blockchain*: mantienen una copia completa de la *blockchain* con todas las transacciones, y pueden verificar todas las transacciones sin dependencia alguna. Necesitan la red para ir actualizando los nuevos bloques y transacciones.

Es el cliente más habitual, pero existe una proliferación importante de clientes más ligeros pensados para dispositivos de poca capacidad, denominados clientes SPV, y que sólo almacenan las cabeceras de los bloques.

2.2. Tipos de clientes Bitcoin

El cliente bitcoin es el programa de usuario final que le permite realizar transacciones en la red Bitcoin. Las funciones básicas que soporta son las siguientes:

- Generación de la clave privada y realización de pagos seguros
- Información sobre el estado de las transacciones y de la red
- Compartición de eventos con otros nodos clientes peer

El cliente inicial, Statoshi creado por Statoshi Nakamoto, ha evolucionado hacia Bitcoin Core. Este cliente ofrece dos modalidades, ambas con las mismas capacidades de gestión, y mecanismos de configuración y funcionamiento:

- Bitcoind: interfaz sólo de comandos
- Bitcoin-Qt: interfaz gráfica

2.3. Las transacciones Bitcoin

Las transacciones son estructuras de datos que codifican la transferencia de valor entre dos participantes de la red bitcoin.
La transacción se envía a la red, mediante cualquier nodo conectado. Si es validada por el nodo, se irá propagando, y le llegará un mensaje de confirmación al remitente. Si la transacción no es válida, el nodo rechaza la transacción, y le enviará un mensaje de rechazo al emisor.

La transacción no formará parte del registro *blockchain* hasta que no sea verificada e incluida en un bloque mediante el proceso de minado.

Las transacciones no tienen una fecha de caducidad, siempre son válidas, pero si no es verificada con un nuevo bloque, sólo estará activa mientras esté en la memoria temporal de los nodos mineros (*pool*). Por contrapartida, el software de *wallet* puede ir retransmitiendo las transacciones hasta conseguir incluirlas en un bloque.

A grandes rasgos, el ciclo de vida de una transacción sería lo mostrado en la siguiente ilustración:

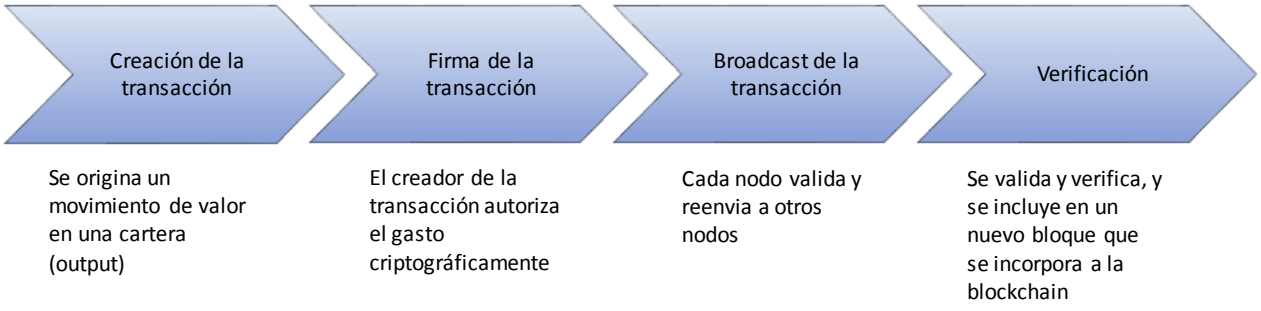


Ilustración 1. Esquema del ciclo de vida de una transacción

2.4. Proceso de conexión a la red Bitcoin

El descubrimiento de la red para un nodo se produce cuando un nodo arranca: al menos tiene que descubrir un nodo existente, y conectarse a él.

Establece una conexión TCP por el puerto 8333, e inicia un proceso de *handshake*, intercambiado la siguiente información:

- Versión de protocolo
- Listado de servicios locales soportados por el nodo
- La hora actual
- Dirección del nodo remoto
- Dirección propia
- Versión de software del nodo
- *Bestheight*: la longitud del *blockchain* de este nodo

A continuación, el nodo receptor envía un mensaje de "*verack*", completándose el proceso de *handshake.*

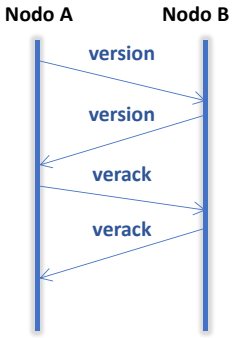


Ilustración 2. Flujo de intercambio de mensajes al inicio de conexión a la red

En el cliente *Bitcoin Cor*e están configurada una lista de "nodos semilla" que se utilizan para las primeras conexiones. Una vez establecida la conexión, los nodos propagan la dirección del nuevo nodo; a su vez, el nuevo nodo puede solicitar la lista de direcciones de los otros nodos. Bitcoin Core preserva un registro de los nodos descubiertos.

Si no hay tráfico en una conexión, los nodos enviarán mensajes periódicos para mantenerla. Si un nodo no comunica durante más de 90 minutos, se determina que no hay conexión, y se busca un nuevo *peer*.

En el caso de los nodos que son "*full node*", en la fase de arranque éste sólo contiene el bloque inicial o "genesis": solicita a los diferentes nodos todos los bloques necesarios para completar el *blockchain,* en un flujo que procura balancear la carga entre los nodos fuente (intercambio de inventario).

2.5. Tipos de mensajes en la red Bitcoin

Como ya hemos expuesto con anterioridad, en la red Bitcoin se intercambian mensajes asociados a las diferentes actividades que experimenta:

- Mensajes de establecimiento y mantenimiento del canal de comunicación entre nodos
- Mensajes asociados a transacciones
- Mensajes asociados a la minería de bloques

Sin pretender ser exhaustivos en los tipos y usos de los diferentes mensajes, los principalmente utilizados serían los indicados en el siguiente cuadro [5]:

Establecimiento y mantenimiento del canal de comunicación	- **addr**: para solicitar las direcciones y números de puertos de otros peers de la red - **version**: mensaje intercambiado entre nodos en el inicio de un proceso de conexión (handshake) - **verack**: mensaje intercambiado entre nodos en un proceso de conexión, cuando ésta es confirmada y se establece - **ping/pong**: son los mensajes intercambiados para comprobar que la conexión entre peers está activa
Mensajes asociados a transacciones	- **inv**: mensaje comunicando una transacción - **get data**: mensaje de respuesta a inv - **tx**: mensaje de respuesta a get data
Mensajes asociados a la minería de bloques	- Se utilizan los mensajes de **block, inv, getdata, getheaders, y getblocktemplate**, **getblocks** en función de la situación - Inv, getdata, block y getblocks se utilizan también cuando un nodo se conecta y tiene que conseguir el blockchain completo de sus peers

En cuanto a los mensajes asociados a transacciones, como ya se ha señalado anteriormente, una transacción es reenviada por un nodo sólo en el caso que ésta es verificada.
Cuando un nodo recibe una transacción de uno de sus vecinos, éste envía a todos sus vecinos un mensaje INV conteniendo el hash de la transacción. El nodo receptor comprueba si el hash de la transacción ha sido enviado anteriormente; si no ha sido enviado anteriormente, el nodo receptor solicita la transacción mediante un mensaje GETDATA. En respuesta, el nodo correspondiente envía los datos de la transacción.

Un nodo no envía un mensaje INV a todos los vecinos disponibles, si no sólo a un número aleatorio, en un periodo de 100ms. En consecuencia, el tiempo total requerido para propagar una transacción depende del número total de nodos vecinos [6].

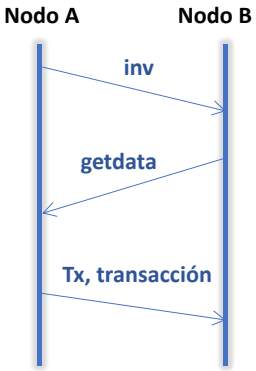


Ilustración 3. Flujo de intercambio de mensajes en una transacción

2.6. Modalidades de funcionamiento de la red Bitcoin

Existen diferentes modos de ejecución de un nodo Bitcoin Core, en función del entorno de red en el que se quiera trabajar:

- **Red Mainet**

Es la red Bitcoin real. El puerto de escucha es el 8333, y el de conexión RPC, el 8332.

- **Red Testnet**

Es una red de pruebas para programadores y pruebas en general. Opera de la misma forma que la red Mainet, pero con una moneda diferente sin valor, y con unos algoritmos de minado mucho más rápidos que en la red real.
El puerto de escucha es el 18333, y el de conexión RPC, el 18332.

- **Red Regtest**

Es una red de pruebas, con la misma finalidad que Testnet. Ofrece la ventaja que el proceso de minado se puede controlar, de forma que permite por ejemplo trabajar con nodos de una forma más aislada.

3. Métricas de interés para el estudio

Una vez comprendido el funcionamiento básico de la red Bitcoin, e identificados los flujos de información más relevantes, podemos establecer una serie de métricas e indicadores interesantes para su estudio:

Grupo	Métrica	Indicador
Inicio de nodo en la red	Número de peers iniciales	Número medio de peers
	Tiempo necesario para propagar dirección en toda la red	Tiempo medio de propagación
Régimen permanente de la red	Número de peers	Número medio de peers
	Tiempo de transmisión de baja de un nodo en la red	Tiempo medio de propagación baja
	Errores de conexión	
	Tiempo de nodo activo/inactivo	Tiempo medio de activo/inactivo
	Tiempo de comunicación con peers	Latencia de la red
	Tiempo de confirmación de las transacciones en recepción	Tiempo medio de transacción

Transacciones		
	Número de nodos que atraviesa una transacción hasta destino	Número medio de saltos Nodos más transitados vs nodos menos transitados
	Transacciones reenviadas	Número medio de reenvios
	Tiempo de validación de las transacciones formando un bloque	Tiempo medio de validación
	Bloques reenviados	Número medio de reenvios

No todos estos elementos pueden ser no obstante objeto de un estudio como el presente, de forma que es necesario focalizarse en alguno de ellos.

La propagación de las transacciones, ofrece un especial interés dada la característica antes expuesta que el broadcast de una transacción no se ejecuta de forma extensiva a todos los nodos conectados al nodo origen, si no por bloques de nodos, en función de un número aleatorio [6].

Un método de referencia a aplicar en el caso de este proyecto sería calcular la suma de las desviaciones de los tiempos de recepción de un mismo mensaje entre los diferentes nodos.
Para obtener un resultado significativo, el número de nodos e iteraciones tendría que ser importante [6].

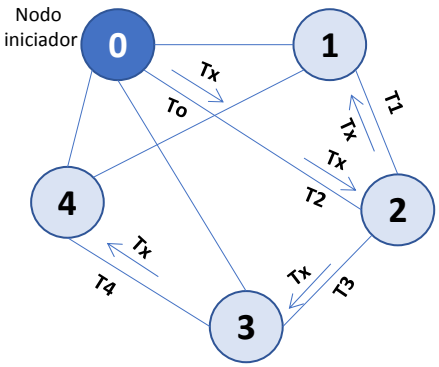


Ilustración 4. Esquema del análisis de tiempos de propagación de una trasacción

4. Plataforma de pruebas

La plataforma se basará fundamentalmente en herramientas existentes, cumpliendo en su conjunto las siguientes funciones:

- Infraestructura de procesamiento / computacional
- Captación de métricas sobre el comportamiento de la red y de los nodos
- Formación de nodos de la red Bitcoin
- Análisis y presentación de la información asociada a las métricas

A continuación, se describen los diferentes elementos seleccionados inicialmente para formar parte de la plataforma.

4.1. Arquitectura global de la plataforma

La arquitectura teórica a implementar se basa en cuatro bloques funcionales:

1. Infraestructura basada en contenedores y construida mediante el entorno btc_testbed, basado en Python
2. Nodos bitcoin funcionando con Bitcoind
3. Análisis de la red basada en StatsD/Statoshi
4. Análisis y presentación de datos con Graphite y Grafana

Los bloques 1, 2, 3 se provisionan en una infraestructura de dockers, i el bloque 4 de análisis y presentación se ubican directamente en la máquina host que también aloja a los dockers.

Así pues, la solución presentaría el siguiente esquema:

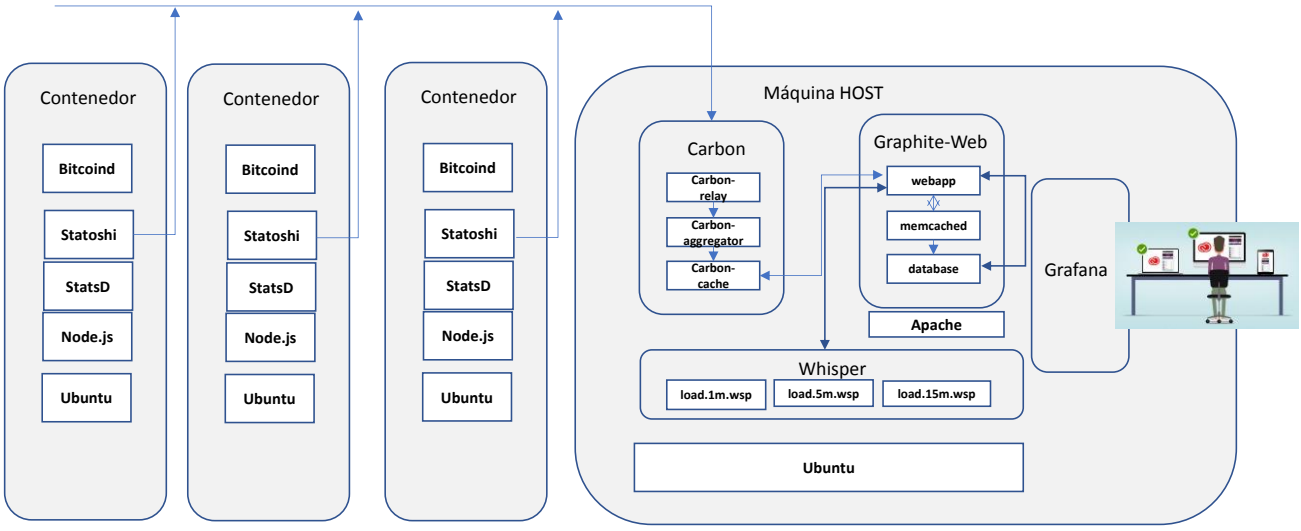


Ilustración 5. Arquitectura teórica de la solución

4.2. Infraestructura computacional

Seleccionamos infraestructura basada en *dockers* (www.dockers.com), por los siguientes motivos:

- Es flexible, escalable
- Multiplataforma
- La mayoría de productos y herramientas disponen de imágenes para *dockers*

4.3. Tratamiento de las métricas

En este caso, la herramienta propuesta es **Statoshi** [7]. Es un sistema de métricas que complementa las existentes con **StatsD**, orientadas a la gestión de la red Bitcoin. StatsD, es un daemon de red de Etsy que funciona sobre *node.js*. Se puede integrar con diferentes visualizadores y herramientas de medida de métricas.

Es un sistema de medida en tiempo real, y presenta estos principales grupos de métricas:

- **bandwidth** – relacionadas con el uso del ancho de banda por parte de los nodos
- **blocks** – relacionadas con las propiedades de un bloque individual
- **message** – relacionadas con mensajes p2p procesados por el nodo
- **misbehavior** – relacionadas con comportamientos extraños de los nodos
- **peers** – agrega propiedades de los *peers* a los que está conectado el nodo
- **transactions** – agrega propiedades de las transacciones que ha procesado el nodo
- **utxoset** - propiedades del actual conjunto *UTXO (upsent transaction outputs)*

Algunas de las funciones que podremos comprobar serán como las indicadas en la Ilustración 6 [8].

p2p_compactblocks.py	[tests] Rename p2p_* functional tests.
p2p_disconnect_ban.py	[tests] Rename p2p_* functional tests.
p2p_feefilter.py	[tests] Rename p2p_* functional tests.
p2p_fingerprint.py	Fix a-vs-an typos
p2p_invalid_block.py	[tests] Rename p2p_* functional tests.
p2p_invalid_tx.py	[tests] Change invalidtxrequest to use BitcoinTestFramework
p2p_leak.py	[tests] Rename p2p_* functional tests.
p2p_mempool.py	[tests] Rename p2p_* functional tests.
p2p_node_network_limited.py	[QA] add NODE_NETWORK_LIMITED address relay and sync test
p2p_segwit.py	[tests] Rename p2p_* functional tests.
p2p_sendheaders.py	[tests] Rename p2p_* functional tests.
p2p_timeouts.py	[tests] Rename p2p_* functional tests.
p2p_unrequested_blocks.py	[tests] Rename p2p_* functional tests.

Ilustración 6. Funciones que implementan métricas Statoshi

A grandes rasgos, el funcionamiento de *Statoshi* requiere la instalación de los siguientes componentes:

- Sistema operativo Ubuntu
- Node.js
- StatsD
- Apache http server
- Graphite
- Statoshi
- Compilación

Cabe destacar que si el nodo bitcoin no está en la misma máquina que *StatsD*, hace falta indicarlo en los ficheros de configuración de *Statoshi* (/statoshi/src/statsd_client.h).

Las tipologías de métricas tanto de Statsd como de Statoshi se basan en tres modelos de datos:

- **counters –** es el tipo más básico de datos. Cuentan las veces que un evento se produce por segundo. Normalmente el valor se promedia para un minuto.
- **timers -** su objeto es medir la duración de un evento. Statsd calcula para un periodo determinado, el valor mínimo, la media, el percentil 90, y el máximo, así como las veces que se produce un valor
- **gauges –** son valores constantes y absolutos que no se promedian ni modifican por parte de statsd

4.4. Entorno de análisis y presentación

Siguiendo las recomendaciones de la plataforma *Statoshi*, para este bloque seleccionamos **Graphite** [9]: es una herramienta *opensource* de monitorización para visualizar métricas previamente recolectadas. Está conformada por tres componentes:

- Colector de métricas. Permite incorporar diferentes tipos de colectores de métricas, entre ellos, **StatsD.**
- El *listener*. En este caso es **Carbon.** Escucha las métricas que se reciben del colector, y se dejan en disco. Permite escalar en diferentes *daemons* para registrar altos volúmenes de métricas.
- La base de datos de almacenaje. En este caso es **Whisper**.

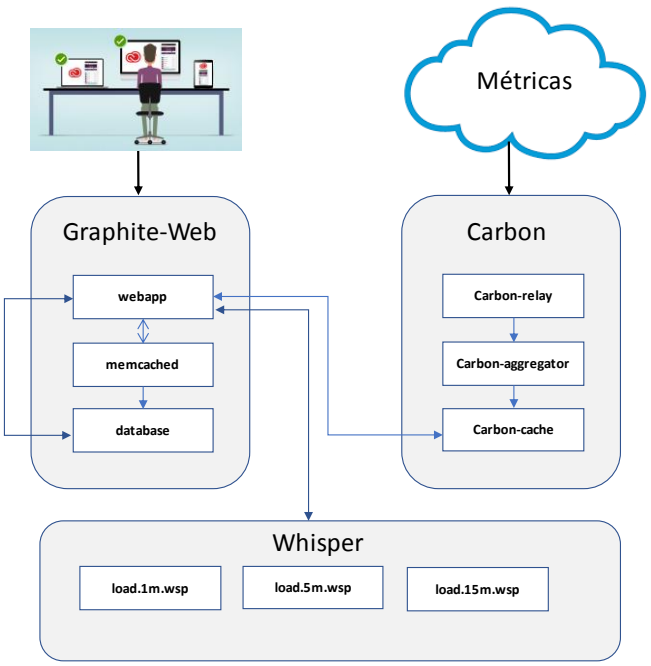


Ilustración 7. Componentes básicos de Graphite

En cuanto a **Grafana**, es una herramienta para construir cuadros de mando, a partir de los resultados obtenidos con Graphite.

4.5. Formación de los nodos bitcoin

El entorno seleccionado se basa en el esqueleto facilitado para la ejecución del proyecto. Presenta las siguientes características:

- Se basa en infraestructura de dockers
- Facilita funciones básicas para la creación de contenedores
- Facilita funciones básicas para la revisión de los contenedores
- Facilita esquemas básicos de redes de contenedores para alojar los correspondientes nodos bitcoin

En todo caso, deberemos instalar el cliente **Bitcoind** para desplegar el servicio Bitcoin en los diferentes nodos.

Las funciones existentes en el esquema son las siguientes:

Fichero rpc_utils.py:

- rpc_getinfo: recupera información sobre la conexión RPC de un contenedor que alberga un nodo bitcoin
- rpc_test_connection: comprueba la conexión sobre la conexión RPC de un contenedor que alberga un nodo bitcoin
- rpc_getpeerinfo: envía un getpeerinfo a un contenedor que alberga un nodo bitcoin

- rpcp_get_peer_ips: devuelve la lista de IPs que tiene el nodo como peers
- rpc_create_connection: crea una conexión entre dos nodos bitcoind
- rpc_call: envía una rpc call a un contenedor determinado
- rpc_call_to_all: envía a call a todos los contenedores con un prefijo determinado
- rpc_get_network_topology: consigue la topología de red. Relación de contenedores, las direcciones IPs propias y de sus peers.

Fichero Docker_utils.py:

- get_containers_names: devuelve el nombre de los contenedores
- count_containers: cuenta el número de contenedores existentes y su prefijo identificativo
- remove_containers_by_name: borra un contenedor determinado
- remove_containers: borra todos los contenedores identificados según un prefijo
- get_ip_by_containers_name: devuelve la IP de un contenedor
- is_valid_ip: comprueba si una cadena es una IP válida
- get_ip_by_unknown: devuelve la IP de un contenedor en función de su nombre o IP
- run_new_node: crea un nuevo contenedor
- run_new_nodes: crea n contenedores
- create_network: crea una red de contenedores
- get_container_name_by_ip: devuelve la IP de un contenedor a partir de su nombre

Run_escenarios.py:

- create_basic_escenario: crea una red básica de dos nodos y una conexión entre ambos
- create_scenario_from_graph: crea una red según la topología indicada con un grafo (parámetro de entrada)
- create_scenario_from_graph_file: crea una red según la topología indicada con un fichero graphml (parámetro de entrada)
- create_scenario_from_er_graph: crea una red aleatoria
- run_scenario_vicl: crea un escenario basado en un fichero de grafos, muestra información de los diferentes nodos. Simula un proceso de minado y además, simula la desconexión de dos nodos.
- docker_setup: crea un contenedor, y opcionalmente: la imagen del docker, la red, y borra los contenedores existentes

Se deduce pues, que es un sistema flexible y que permite ampliar funcionalidades.

5. Implementación de la plataforma

A continuación se describe el proceso de instalación y puesta en marcha de la plataforma de pruebas, según los diferentes componentes.

5.1. Máquina host

En la máquina host, se aloja todo el entorno, de acuerdo con la siguiente configuración de base:

- Sistema operativo equipo host: Ubuntu 14.04,
- Instalación de docker: docker-CE
- Python 2.7.6

5.2. Plataforma btc_testbed – configuración inicial

La configuración inicial de la plataforma permite crear diferentes escenarios de nodos bitcoin en modalidad regtest. Su instalación se realiza en la máquina host, y para el funcionamiento general, se cargan las siguientes librerías, previa instalación de Python-pip:

- Pip intall docker
- Pip install python-bitcoinrpc
- Pip install logging
- Pip install ecdsa
- Pip install base58
- Pip install networkx

El funcionamiento de run_scenarios requiere los siguientes paquetes:

- Pip install matplotlib.pylot
- Apt install pkg-config
- Apt install libpng12-dev
- Apt install libfreetype6-dev
- Apt install Python-dev
- Pip install matplotlib

La plataforma btc_testbed presenta dos ámbitos bien diferenciados:

Entorno de configuración, en el que se hallan los ficheros clave que definen los parámetros generales:

- conf.py: define todas las características globales de la red
- bitcoin.conf: define cómo será la imagen ejecutable bitcoin
- Dockfile: ejecuta los comandos necesarios para crear la imagen bitcoin, identificada como DOCK_IMAGE_NAME

Entorno de ejecución de la red, en el que se procede a la construcción del misma, contando con los siguientes ficheros en lenguaje Python:bx run_scenarios

- docker_utils
- rpc_utils

La plataforma se inicia ejecutando el fichero run_scenarios.[1]py, que permite seleccionar diferentes configuraciones de redes bitcoin:

- create_basic_escenario: crea una red básica de dos nodos y una conexión entre ambos
- create_scenario_from_graph: crea una red según la topología indicada con un grafo (parámetro de entrada)
- create_scenario_from_graph_file: crea una red según la topología indicada con un fichero graphml (parámetro de entrada)
- create_scenario_from_er_graph: crea una red aleatoria
- run_scenario_vicl: crea un escenario basado en un fichero de grafos, muestra información de los diferentes nodos. Simula un proceso de minado y además, simula la desconexión de dos nodos.

La función docker_setup, incluida en run_scenarios.py, es la que permite la creación de la imagen (DOCK_IMAGE_NAME), a partir del fichero Dockerfile.

En estos diferentes escenarios se hacen llamadas a funciones del fichero docker_utils.py para la ejecución de comandos sobre los contenedores de la red.

Una de las funciones clave de docker_utils.py es la run_new_node, que permite crear los contenedores deseados con la imagen que se seleccione.

A su vez, desde las funciones definidas en docker_utils.py se hacen llamadas a las funciones del fichero rpc_utils.py, que contiene las llamadas rpc necesarias para gestionar los nodos bitcoin.

A continuación se representan los elementos principales de la red btc_testbed, y cómo se relacionan entre sí:

[1] En la implementación de este proyecto, las sentencias tipo blocks_0_after = rpc_getinfo(client, "btc_n0")["blocks"], generaban error, y se ha suprimido; [“blocks"].

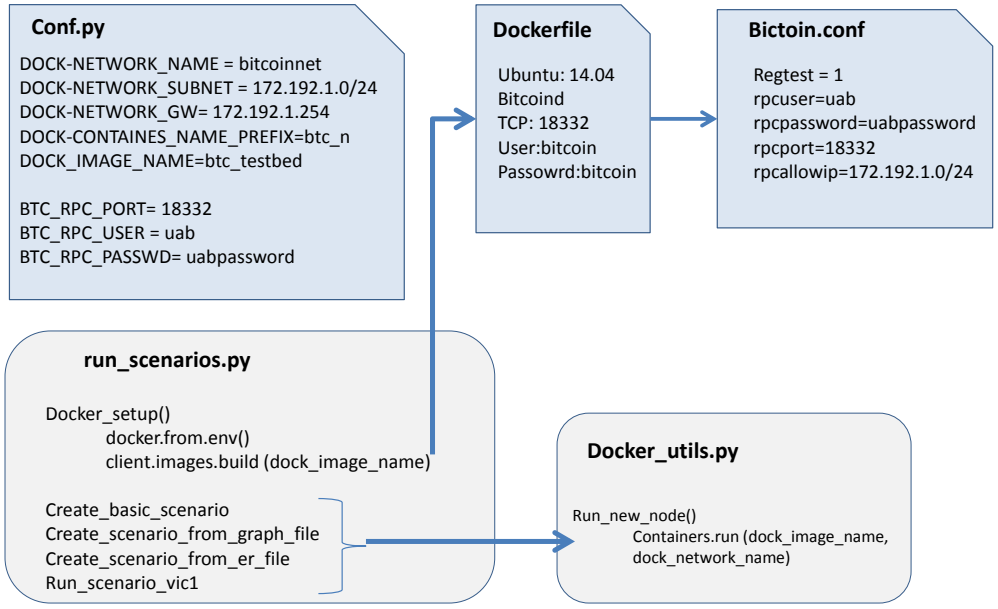


Ilustración 8. Elementos principales de btc_testbed

5.3. Plataforma btc_testbed – configuración modificada

Para poder desplegar el sistema de métricas y de análisis de indicadores, nos servimos de las siguientes herramientas:

- Análisis de la red basada en StatsD/Statoshi
- Análisis y presentación de datos con Graphite y Grafana

Las herramientas StatsD/Statoshi, dedicadas a la recopilación de métricas, serán desplegadas dentro de la propia imagen que habilita la ejecución de los nodos bitcoin, y Graphite/Grafana se instalarán directamente en la máquina host.
Para el funcionamiento de Graphite/Grafana en la máquina host, será necesario un servidor web Apache activo que publique ambas.

En consecuencia, para desplegar StatsD/Statoshi:

- la imagen de docker a construir agregará la configuración de bicoin existente, con statsd y statoshi, comportando la modificación del fichero Dockerfile
- la imagen resultante será la que se desplegará en los contenedores construidos desde run_scenarios.py

5.3.1. Imagen del docker a construir

Para la instalación de statsd/statoshi nos servimos de statoshi-lite [10][11], que permite el despliegue completo en docker de: bitcoind, statsd, statoshi, el servidor http apache, Graphite y Grafana. No obstante, en nuestro caso limitamos la instalación de la imagen a bitcoind, statsd y statoshi.

Para el correcto funcionamiento sobre el entorno de trabajo, se realizan los siguientes cambios sobre el fichero Dockerfile:

- Fijar el sistema operativo a Ubuntu 16.04. Cualquier versión inferior, o parametrizar el sistema operativo como *FROM ubuntu:latest*, genera problemas de procesado del fichero en la parte de npm y node.js
- La instrucción *RUN ln -s ""$(which nodejs)"" /usr/bin/node* genera también problemas. Ha sido substituida por

  RUN update-alternatives –install /usr/bin/node nodejs /usr/bin/nodejs 100

- Se incorpora a Dokcerfile el fichero bitcoin.conf propio de btc_testbed:
  - Eliminamos:

    RUN wget "http://yabtcn.info/statoshi/bitcoin.conf.example" -O /home/statoshi/.bitcoin/bitcoin.conf >/dev/null 2>&1

  - Incorporamos:

    ADD –chown=statoshi:statoshi bitcoin.conf /home/statoshi/.bitcoin/

El fichero bitcoin.conf preserva la misma definición original de btc_testbed:

dnsseed = 0	→ No utiliza un servidor DNS de referencia para el descubrimiento de la red
regtest = 1	→ Activación de los nodos en la red Regtest
rpcport= 18333	→ Puertos utilizados en la red Regtest
rpcallowip=172.192.1.0/25	→ Permitimos conexiones RPC de direcciones del mismo rango de la red

Finalmente, el fichero Dockerfile presenta las siguientes fases de construcción de la imagen:

1. Sistema operativo, paquetes y librerías necesarios de forma genérica
2. Creación de la estructura de directorios y del usuario de statoshi, que será su propietario
3. Descarga de los siguientes ficheros, que permitirán que la ejecución de los procesos sea permanente:
   - Stats.check.sh (arranque de StatsD)
   - Systemmetricsd.check.sh (activación del daemon de sistema de métricas de Statoshi)
4. Descarga de bitcoin-utils
5. Copia del fichero bitcoin.conf al directorio correspondiente del contenedor
6. Instalación de statoshi (core bitcoin): descarga de los ficheros y compilación.
7. Instalación y activación de statsd, requiriendo previamente instalación de nodejs, npm, forever
8. Descarga y configuración de los programas de cron para mantener siempre en ejecución de stats.check.sh y systememtricsd.check.sh

<u>Particularidades a tener en cuenta en algunas configuraciones</u>:

- Configuración de la conexión entre StatsD y Graphite. Stats.check.sh: arranca statsd a partir del fichero /opt/statsd/config.js

  En este fichero hay que configurar los parámetros de envío de statsd de las diferentes métricas:

```
graphitePort: 2003                  → Puerto TCP de envío de métricas a graphite
graphiteHost: "192.168.1.44"        → Dirección IP del host que aloja a graphite
port: 8125                          → Puerto UDP que utiliza Statsd para recoger métricas
backends: [ "./backends/graphite"]  → Especificamos que el backend es graphite
```

- Configuración del puerto de comunicación con la red bitcoin. Al trabajar en la red tesbed, especificamos que el puerto a utilizar es el 18332:

  EXPOSE 18332

5.3.2. Configuración de Graphite

Graphite presenta varias partes importantes a tener en cuenta en su instalación [12]:

- La base de datos necesaria para su publicación web
- El fichero de configuración para la publicación web
- El fichero de configuración de carbon
- El fichero de configuración del almacenamiento de métricas

A continuación concretamos cuáles son los aspectos clave para su configuración:

- Base de datos publicación web

La base de datos para la publicación web seleccionada es postgreSQL (graphite). En ella Graphite almacena información como la configuración de gráficos y cuadros de mando, y cuentas de usuario.
Se configura un usuario "graphite"

- Fichero de configuración para la publicación web (local_settings.py)

Se configuran algunos parámetros, como TIME_ZONE, siendo relevante la configuración correcta de la base de datos, de acuerdo con lo que se haya definido en el punto anterior:

```
DATABASES = {
        'default': {
                'NAME': 'graphite',
                'ENGINE': 'django.db.backends.postgresql_psycopg2',
                'USER': 'graphite',
                'PASSWORD': 'graphite',
                'HOST': '127.0.0.1',            → máquina que aloja graphite (localhost)
                'PORT': ''
                }
        }
```

- Fichero de configuración de carbon (carbon.conf)

Toda la configuración que viene por defecto es válida, y sólo hay que cambiar una variable:

ENABLE_LOGROTATION = True

- Fichero de configuración del almacenamiento de métricas (storage-schemas.conf)

En él se especifica los patrones de métricas a tratar, y cuánta frecuencia y por cuánto tiempo carbon retendrá las métricas.

En el caso que nos ocupa, nos interesa procesar las métricas que correspondan con el patrón Stats/Statsd. Hacemos tres tipos de captura, con frecuencias y periodos de retención diferentes:

```
[carbon]
pattern = ^carbon\.
retentions = 10:2160,60:10080,600:262974

[stats]
priority = 110
pattern = ^stats\\..*
retentions = 10:2160,60:10080,600:262974

[default]
pattern = .*
retentions = 10:2160,60:10080,600:262974
```

Para la ejecución de graphite, primero habilitamos la conexión con la base de datos, y después, ejecutamos graphite:

```
graphite-manage syncdb
service carbon-cache start
```

5.3.3. Configuración del servidor Apache

El servidor presentará dos ubicaciones, una para Graphite, y otra para Graphana.
Para ello, configuramos que Graphite trabaje en el puerto TCP 8080, y Grafana en el TCP: 80:

- Con el paquete de Graphite ya disponemos del fichero de configuración (apache2-graphite.conf), al que sólo habrá que modificar:

  <VirtualHost *:8080>
- Modificamos el fichero ports.conf:

  Listen 80
  Listen 8080

5.3.4. Configuración de Grafana

Grafana presenta estos elementos relevantes para su correcta configuración:

- La base de datos necesaria para su publicación web
- El fichero de configuración de Grafana
- La configuración proxy del servidor Apache
- La configuración en Grafana de la conexión con Graphite

- Base de datos publicación web

La base de datos para la publicación web seleccionada es postgreSQL (grafana). En ella Graphite almacena información como la configuración de gráficos y cuadros de mando, y cuentas de usuario.

Se configura un usuario “graphite”.

- Fichero de configuración de Grafana (grafana.ini)

El aspecto más relevante a configurar, respecto a los valores por defecto, es el acceso a la base de datos creada anteriormente:

```
[database]
# Either "mysql", "postgres" or "sqlite3", it's your choice
type = postgres
host = 127.0.0.1:5432
name = grafana
user = graphite
password = graphiteuserpassword
```

- Configuración proxy del servidor Apache

Grafana dispone de un fichero de configuración de su site (apache2-grafana.conf), en el que hay que especificar que se active el Proxy Pass y Reverse. El puerto utilizado entre Grafana y Graphite es el TCP 3030.

- La configuración en Grafana de la conexión con Graphite

Ya en la consola de Grafana, se da de alta como fuente de datos (DataSource) Statsd, y posteriormente se crean los dashboards correspondientes.

El servicio de Grafana arranca con: service grafana-server start.

Para la configuración inicial de los paneles de Grafana, exportamos los ficheros JSON que publica http://statoshi.info/, en concreto los referidos al dasboard de mensajes P2P.
El motivo para seleccionar este grupo de indicadores es que en el inicio de la red son las métricas más inmediatas de detectar.
Los mensajes monitorizados son: ping, pong, addr, inv, getData, tx, y se corresponden con las siguientes métricas de statsd/statoshi:

- stats.message.received.ping
- stats.message.received.pong
- stats.message.received.addr
- stats.message.received.inv
- stats.message.received.getdata
- stats.message.received.tx

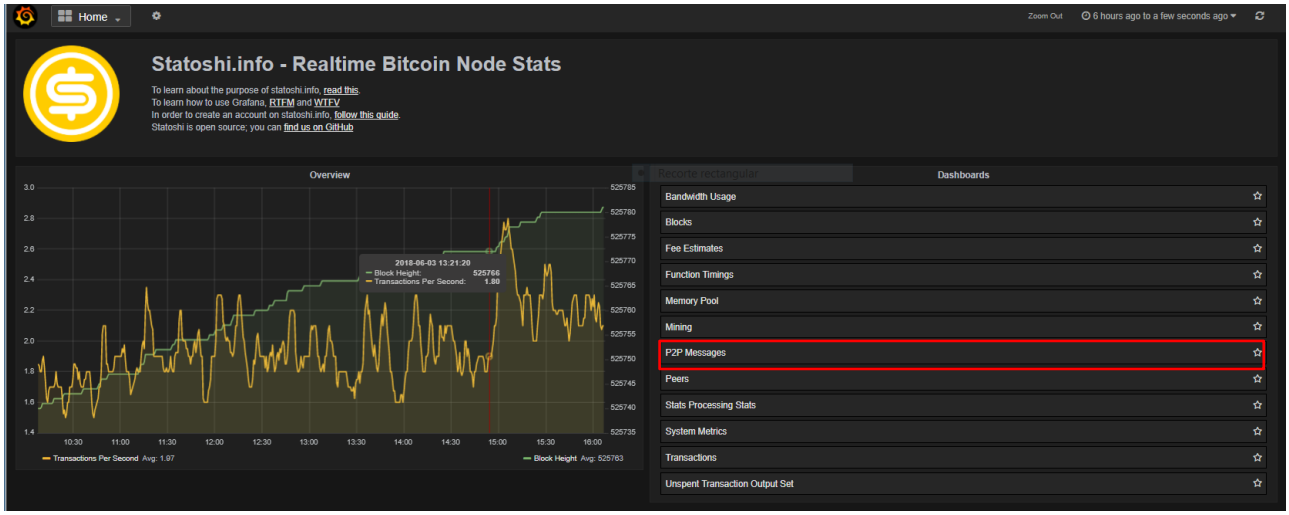


Ilustración 9. Selección de cuadro de mando y métricas en Statoshi.info

En el siguiente esquema se representan los principales elementos y dependencias de la plataforma implementada:

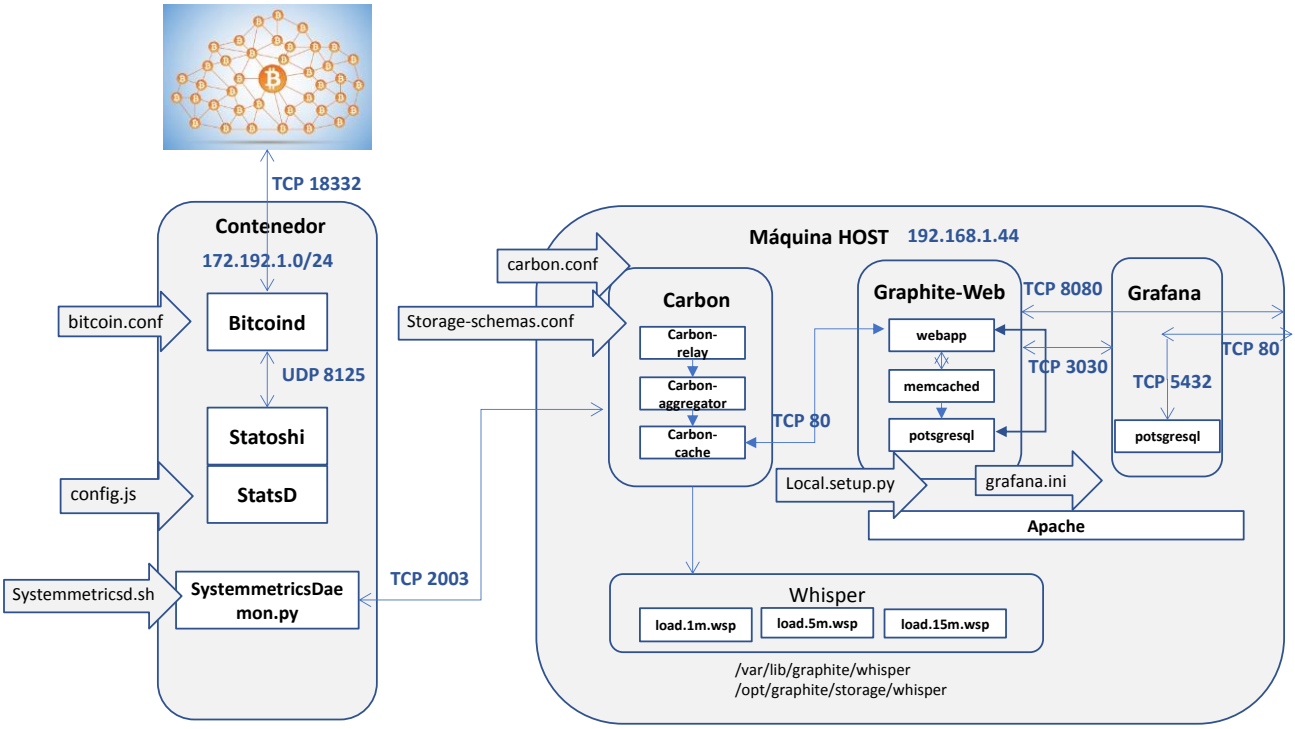


Ilustración 10. Esquema global de configuración de la plataforma

6. Ejecución de la plataforma

Iniciamos la actividad de los nodos Bitcoin mediante la ejecución de run_scenarios.py, y la selección del escenario run_scenario_vic1 (client).
Recordemos que este escenario creaba cuatro nodos a partir de un grafo.

Comprobamos la activación de los cuatro nodos desde el punto de vista de docker:

```
root@cristina-VPCEH2M0E:/home/cristina/Escritorio/projecte/btc_testbed# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS                                   NAMES
ca957d32706c        btc_testbed         "bitcoind -debug"   6 minutes ago       Up 6 minutes        2003/tcp, 0.0.0.0:22003->18332/tcp      btc_n
6571f76bfd19        btc_testbed         "bitcoind -debug"   6 minutes ago       Up 6 minutes        2003/tcp, 0.0.0.0:22002->18332/tcp      btc_n2
6c5b53d3c0b8        btc_testbed         "bitcoind -debug"   6 minutes ago       Up 6 minutes        2003/tcp, 0.0.0.0:22001->18332/tcp      btc_n1
2d3d398e2ba4        btc_testbed         "bitcoind -debug"   6 minutes ago       Up 6 minutes        2003/tcp, 0.0.0.0:22000->18332/tcp      btc_n0
root@cristina-VPCEH2M0E:/home/cristina/Escritorio/projecte/btc_testbed#
```

Ilustración 11. Comprobación de los dockers de los diferentes nodos

Asimismo, comprobamos la actividad de uno de los nodos:

```
root@cristina-VPCEH2M0E:/home/cristina/Escritorio/projecte/btc_testbed# bitcoin-cli -rpcconnect=172.192.1.3 getwalletinfo
{
  "walletname": "wallet.dat",
  "walletversion": 159900,
  "balance": 50.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 3650.00000000,
  "txcount": 101,
  "keypoololdest": 1528017733,
  "keypoolsize": 999,
  "keypoolsize_hd_internal": 1000,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "e8108a33a9300edfce248a0eef459c38f91c080b"
}
root@cristina-VPCEH2M0E:/home/cristina/Escritorio/projecte/btc_testbed#
```

Ilustración 12. Comprobación de uno de los nodos

También comprobamos que tanto Graphite como Grafana reciben actividad de las comunicaciones que realizan los nodos, basadas en ping y pong:

Ilustración 13. Monitorización en Grafana de los mensajes ping/pong de la red

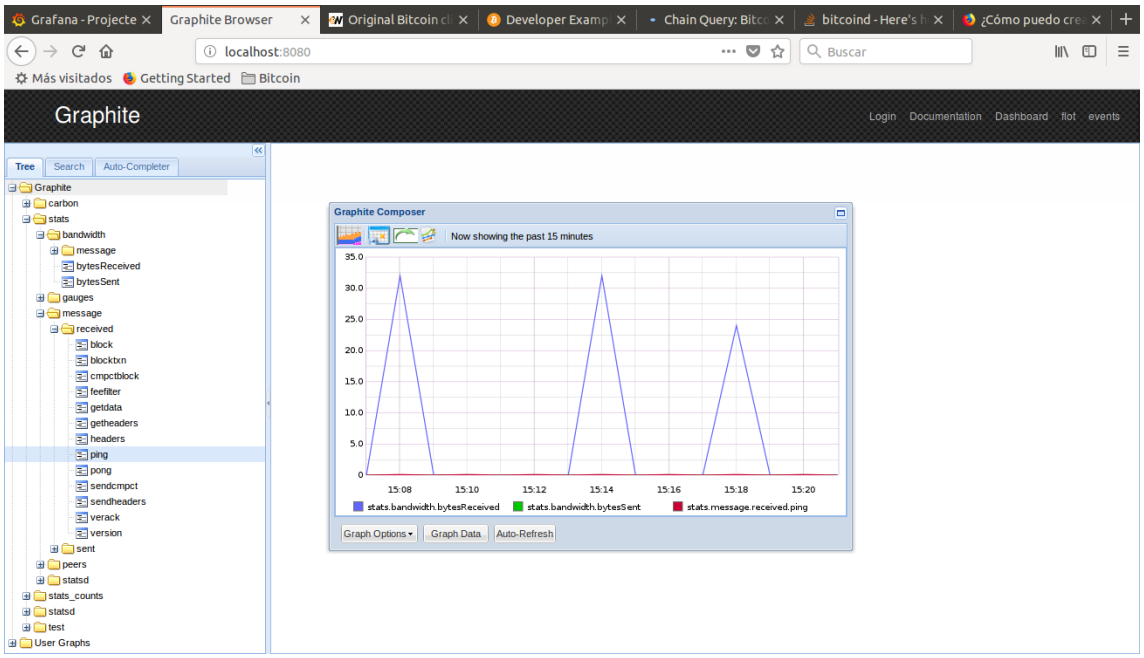


Ilustración 14. Monitorización en Graphite de los mensajes ping de la red

Al tratarse de una red tipo regtest, la actividad de la misma se controla mediante una capa de aplicación de programador.

De esta forma, es necesario ejecutar comandos que por ejemplo, permitan una transacción económica, utilizando órdenes RPC como sendtoaddress, createrawtransaction, signrawtransaction o sendrawtransaction [13][14].

7. Conclusiones

El proyecto realizado ha permitido incorporar al entorno btc_testbed, conformado con funciones Python que despliegan nodos bitcoin, funciones de análisis y representación de métricas basados en Statsd/Statoshi, y Graphana/Grafite.

La explicación detallada de su implementación permite entender en profundidad cómo funcionan e interactúan los diferentes módulos de la plataforma, y cuáles son los elementos clave que permiten su configuración y puesta en marcha. Para ello, ha sido necesario utilizar diversas fuentes de información.

Asimismo, el objetivo de disponer de una plataforma se ha compatibilizado con un estudio de los flujos de información que se producen en la red Bitcoin. Sin él no sería viable identificar y definir métricas e indicadores que analicen su comportamiento.

Los próximos pasos que darían continuidad al trabajo realizado serían los siguientes:

- Relacionados con la plataforma

Estandarizar la configuración de la parte de host, en la que alojamos btc_testbed, docker, Grafana y Graphite, creando un docker específico.

- Relacionados con el análisis de la red Bitcoin

Desarrollar herramientas en Python con APIS NodeJS que generen diferentes tipos de tráfico en la red btc_testnet. En especial, enfocando los esfuerzos en el comportamiento y tiempo de propagación de las transacciones, siguiendo los principios del estudio *Bitcoin Network Measurements for Simulation Validation and Parameterisation, de la Universidade Porstmouth* [6] (o más recientes).

Creación de una base de datos que permita el procesado de los mensajes para calcular, por ejemplo, el tiempo transcurrido en una propagación total.

Después del presente estudio, se concluye que el mejor método para un análisis de comportamiento de los flujos de información, es un uso de métricas basadas en statsd/statoshi combinado con el almacenamiento y procesado de datos en una base de datos.

Finalmente, como conclusión del trabajo expuesto y de los resultados obtenidos, el proyecto consigue una visión de conjunto, tanto del funcionamiento y factores clave de la red Bitcoin, como de las herramientas necesarias para analizarla, así como su implementación.

Sobre la base establecida, quedará para fases posteriores el estudio analítico exhaustivo de los flujos de información tales como las transacciones en la red Bitcoin.

8. Bibliografía

[1]https://en.wikipedia.org/wiki/Bitcoin
[2]https://www.criptonoticias.com/opinion/repaso-historia-bitcoin-nueve-anos-minarse-bloque-genesis/
[3] https://weeraman.com/dissecting-bitcoin-part-2-challenges-57f4b0deb2d4
[4] Andreas M. Antopoulus. Mastering Bitcoin
[5] https://bitcoin.org/en/developer-guide
[6] Muntadher Fadhil; Gareth Owen; Mo Adda. Bitcoin Network Measurements for Simulation Validation and Parameterisation
https://researchportal.port.ac.uk/portal/files/7765835/CSCAN_OA_330.pdf
[7] Jameson Lopp. https://medium.com/@lopp/statoshi-developer-s-guide-241ac9ab9993
[8] https://github.com/jlopp/statoshi/tree/master/test/functional
[9] Chris Davis. Graphite Documentation. Release 1.1.2.
https://media.readthedocs.org/pdf/graphite/latest/graphite.pdf
[10] https://hub.docker.com/r/yabtcn/statoshi-lite/
[11] Javier Pérez Diaz, Plataforma para analizar la red Bitcoin. UOC.
http://openaccess.uoc.edu/webapps/o2/handle/10609/64826
[12] https://www.linode.com/docs/uptime/monitoring/how-to-install-graphite-and-grafana-on-ubuntu-14-04/
[13] https://bitcoin.org/en/developer-examples#transactions
[14] http://chainquery.com/bitcoin-api/