Projecte Final de Carrera
Llicenciatura en Enginyeria Informàtica
Universitat Oberta de Catalunya (UOC)

Exactitud d'una aranya en descobrir la topologia de la xarxa Bitcoin.

(Accuracy of a crawler to determine the Bitcoin's network topology)

Autor: Xavier Reina Virgili <xreinav@uoc.edu>

Supervisió: Cristina Pérez Solà <cperezsola@uoc.edu>

4 de gener 2016

UOC
www.uoc.edu

(pàgina en blanc)

Llicència

Del present document

Del codi font del simulador

D'obres de tercers

Agraïments

A Cristina Pérez Solà, directora del projecte, per les exhaustives i detallades correccions realitzades en el desenvolupament del projecte.

Al Dr. Jordi Herrera-Joancomartí i a Cristina Pérez Solà per haver-me introduït a les tecnologies de Bitcoin i *BlockChain*.

Abstracte

El present projecte final de carrera, adscrit a l'àrea de Seguretat de la llicenciatura en Enginyeria Informàtica de la Universitat Oberta de Catalunya, consisteix en l'estudi de l'exactitud d'una aranya –o *crawler* en anglès– per determinar la topologia de la xarxa distribuïda sobre la que es sosté la moneda criptogràfica Bitcoin.

Les aranyes són components de programari que es dediquen a l'anàlisi d'una xarxa per tal de conèixer els recursos que hi ha disponibles. En el marc d'aquest treball ens centrarem en les aranyes que analitzen la topologia de la xarxa d'iguals de la moneda virtual.

Com que la xarxa que sosté a Bitcoin és distribuïda i descentralitzada, és complex determinar l'eficàcia d'una aranya en aquest context, en el benentès que no es pot conèixer la totalitat de la xarxa (degut a la seva descentralització) pel que no es pot comparar la topologia coneguda per l'aranya.

Per aquest motiu s'empra el BTC-NS, una eina de simulacions discretes que permet emular una xarxa d'iguals Bitcoin. En aquest sentit, gràcies a l'eina es pot conèixer la topologia completa de la xarxa i els nodes que va descobrint l'aranya en la simulació, pel que es poden comparar els dos valors i analitzar el comportament de l'aranya.

Per últim, l'eina de simulació no es troba completament finalitzada, pel que al llarg del projecte es duran a terme les modificacions necessàries per tal que al comportament del simulador, pel que fa a la xarxa d'iguals, s'assembli a Bitcoin Core, un programari que es considera "el client de referència de Bitcoin".

Estructura del document

El document es troba estructurat en 11 capítols que es divideixen en tres grans blocs: l'estudi previ i planificació del projecte (capítols 1 a 4), l'anàlisi del funcionament de la xarxa d'iguals Bitcoin i adaptació del simulador (capítols 5 a 7) i estudi de l'eficàcia de l'aranya, conclusions i millores futures (capítols 8 a 11).

Índex

Control de canvis

Versió	Data	Autor	Descripció
0.1	2015-01-05	X. Reina	Correccions en el mode en que es calcula el retard dels missatges (apartat 7.2.1) i ampliat el resultat de l'estudi.
0.09	2015-01-04	X. Reina	Versió final
0.08	2015-12-18	X. Reina	Reestructuració de l'apartat 7 per exposar les millores a realitzar en el simulador en: problema, anàlisi i solució. Afegit apartat 7.3 i millorat el 7.2.2. Redacció capítol 8. Incorporada retroalimentació consultor versió 0.07.
0.07	2015-12-01	X. Reina	Incorporada retroalimentació consultor versió 0.05. Petites correccions. Afegit l'apartat 5.1 on s'exposa el *handshake* de BTC. Afegit en l'apartat 5 l'agrupament per /16. Correccions i ampliacions en el capítol 6.

Versió	Data	Autor	Descripció
			Afegits continguts en la secció 10, millores.
0.06	2015-10-25	X. Reina	Incorporada retroalimentació consultor versió 0.04. Afegit capítol 7
0.05	2015-10-22	X. Reina	Incorporada retroalimentació consultor versió 0.03. Afegit capítol 6.
0.04	2015-10-05	X. Reina	Afegida capítol 5. Afegit en el preàmbul la llista de bocins de codi i els components de tercers. Afegit annex I, II i III.
0.03	2015-09-20	X. Reina	Incorporada retroalimentació del consultor. Incorporats els capítols 1, 2 i 3 Afegit capítol 4.
0.02	2015-09-06	X. Reina	Escrita la introducció a Bitcoin.
0.01	2015-08-30	X. Reina	Definida la taula de continguts.

Llista de taules

Llista d'imatges

Llista de fragments de codi

Acrònims

BD	Base de dades.
BLOB	De l'anglès *Binary Large Object*. Objecte o variable d'una mida rellevant que està format o emmagatzema un conjunt de bits.
CLOB	De l'anglès *Character Large Object*. Objecte o variable d'una mida rellevant que està format o emmagatzema un conjunt de caràcters.
DNS	De l'anglès *Domain Name System*, el protocol de resolució de noms de domini a adreces IP.
GML	De l'anglès *Graphic Modelling Language* [1].
ICMP	*Internet Control Message Protocol*.
NAT	De l'anglès *Network Address Translation*, el mecanisme de traducció d'adreces.

TCP	De l'anglès *Transmission Control Protocol*, el protocol d'Internet orientat a connexions.
P2P	Xarxa d'iguals, de l'anglès *Peer to Peer*.
P2PKH	Pay to Public Key Hash, un dels sistemes per realitzar transferències entre dos adreces de Bitcoin.
WP	Paquet de treball, de l'anglès *Work Package*.

Taula 1: llista d'acrònims

Definicions

Adreça de bitcoin	Una adreça de Bitcoin és el pseudònim on es poden remetre Bitcoins. En termes pràctics és el resum d'una clau pública generada amb criptografia de corba el·líptica.
Client	En el context del projecte es refereix a una instància d'un programari que permet treballar amb la xarxa Bitcoin. També s'empra com a sinònim de *node*.
Commit	Acció en un sistema de gestió de versions (com Subversion o Git) que consisteix en incorporar continguts en un repositori.
Contracte intel·ligent	En anglès *smart contract*, una tecnologia que permet descriure condicions contractuals mitjançant un llenguatge informàtic per tal que un sistema de còmput pugui assegurar-ne el seu compliment sense intervenció directa humana.
Bitcoin	La tecnologia de la moneda criptogràfica.
Bitcoins	Les *monedes* virtuals de Bitcoin.
Doble despesa	Un dels atacs més rellevants en l'àmbit de les monedes criptogràfiques que consisteix en gastar de forma il·legítima dos cops la mateixa moneda.
Funció resum	Funció matemàtica que davant d'una determinada entrada (per exemple un document) torna una cadena d'una longitud determinada. Exemple d'una funció resum seria SHA-256.[1]
Marcador	Emprem el terme "marcador" com la traducció del terme anglès *flag*.
Node	Un membre de la xarxa d'iguals Bitcoin.
Pila	En anglès *stack*. Estructura de dades que es caracteritza perquè les primeres dades que s'hi introdueixen són les últimes que es recuperen.
Reflexió	La invocació de funcions per reflexió és una característica d'un llenguatge de programació que permet redefinir el seu comportament en temps d'execució. Un exemple de reflexió, seria la invocació d'una funció d'acord al valor d'una cadena de text.
Resum	Resultat d'una funció resum. Els resums es solen considerar, per la seves propietats, com un mètode per identificar *inequívocament* un conjunt de dades.
Sac	De l'anglès *bucket*, una estructura de dades que es caracteritza per actuar com una memòria intermediària que no té els elements ordenats.
Sistema estocàstic	Sistema no determinista i, per tant, que té un comportament totalment o parcialment aleatori.

Taula 2: llista de definicions

[1] Per ampliar informació sobre les funcions resum, com a punt de partida el lector pot adreçar-se a l'article de la Wikipedia: https://en.wikipedia.org/wiki/Hash_function [data de consulta 2015-09-29].

1 Introducció

El present document recull la memòria de les tasques desenvolupades en el projecte final de carrera, de l'àrea de Seguretat de la llicenciatura en Enginyeria Informàtica, consistent en l'estudi de l'exactitud d'una aranya –o *crawler* en anglès– per determinar la topologia de la xarxa distribuïda d'igual a igual Bitcoin.

Les aranyes[2] són eines de programari semi-autònomes que recorren una xarxa informàtica analitzant i recollint la informació que hi ha disponible. En aquest sentit les aranyes de Bitcoin intenten descobrir els diferents nodes que formen part de la xarxa per tenir una visió de la pròpia topologia de la xarxa o obtenir informació diversa.

Per tal de dur a terme l'estudi, en lloc de recórrer i estudiar la xarxa Bitcoin real, es treballarà sobre una simulació; el que permetrà conèixer l'eficàcia d'una aranya en l'exploració dels diferents nodes de la xarxa, ja que es podran comparar d'una banda les dades obtingudes per l'aranya i de l'altra la topologia completa.

1.1 Descripció del treball

La simulació que permetrà obtenir les dades necessàries per l'estudi de l'eficàcia de l'aranya es realitzarà mitjançant BTC-NS, una eina de codi obert que permet emular el comportament de diferents perfils de clients Bitcoin, tant pel que fa el comportament de la pròpia xarxa de distribuïda d'iguals com per la cadena de blocs.

Per tal de realitzar l'estudi objecte del treball prèviament caldrà dur a terme quatre tasques per tal d'assegurar-ne la qualitat: analitzar el funcionament de la xarxa Bitcoin real, determinar i documentar el funcionament del BTC-NS, realitzar adaptacions al programari de simulació per ajustar-ne el comportament i, per últim, realitzar l'estudi.

En primer lloc, per tal de conèixer el funcionament "real" de la xarxa, caldrà estudiar el client de referència de Bitcoin [2] i, per fer-ho, s'analitzarà tant el codi font del programa, escrit en C++, com diversa documentació i bibliografia que permeti conèixer el seu comportament i, per extensió, el comportament "estàndard" de la xarxa.

A continuació es realitzarà una anàlisi del comportament del simulador per assegurar que reprodueix de forma fidedigna el comportament habitual de la xarxa Bitcoin pel que respecta al descobriment dels nodes. En aquest sentit es documentarà el funcionament del BTC-NS, comparant-lo amb el client de referència i identificant aspectes a millorar.

En tercer lloc, en base a l'anterior, s'implementaran en el simulador les característiques necessàries per tal de que el funcionament sigui més fidedigne possible a la xarxa Bitcoin real. El codi que s'afegeixi en el simulador estarà implementat en Python i es realitzaran les proves necessàries per tal d'assegurar que s'integri i funcioni correctament.

Per últim, un cop assegurat que el simulador té un comportament similar al món real, s'executaran diverses simulacions que permetran obtenir dades que s'analitzaran per tal d'estudiar l'eficàcia de l'aranya. Al executar les simulacions caldrà tenir cura en la planificació del tipus de simulació per assegurar que es generi la informació necessària.

Addicionalment a l'anterior, en la redacció de la memòria del projecte, per tal d'introduir al lector al funcionament de Bitcoin i que es pugui situar en context, també s'hi inclourà una introducció als principals conceptes i característiques de la xarxa Bitcoins, com ara la *BlockChain*, els *Blocks*, i el funcionament descentralitzat de la xarxa.

[2] Per ampliar informació sobre les aranyes es pot consultar [29] sobre les aranyes web.

1.2 Objectius i resultats esperats del treball

En la descripció dels treballs s'han identificat cinc tipus de tasques a realitzar en l'àmbit del projecte: (1) preparar i redactar una introducció a les tecnologies Bitcoin, (2) analitzar el comportament de la xarxa, (3) determinar i documentar l'estructura del simulador BTC-NS, (4) realitzar-ne modificacions i (5) estudiar el comportament de l'aranya.

Conseqüentment es defineixen cinc objectius específics a assolir en el marc del projecte:

Codi	Descripció
O1	Redactar una introducció a Bitcoin.
O2	Analitzar el funcionament de Bitcoin a nivell de xarxa.
O3	Documentar el funcionament del BTC-NS, detectant els punts que divergeixen substancialment de la xarxa real.
O4	Implementar les millores detectades en l'O3.
O5	Estudiar el comportament de l'aranya.

Taula 3: objectius del projecte

Aquests objectius comportaran els següents resultats:

Codi	Descripció
R1	Documentació d'introducció a les tecnologies Bitcoin.
R2	Descripció del funcionament de la xarxa de d'iguals
R3	Anàlisi del funcionament del simulador a nivell de xarxa.
R4	Resultats de l'estudi del comportament de l'aranya.
R5	Memòria final del projecte.

Taula 4: resultats del projecte.

Addicionalment, les millores que es puguin introduir en el simulador s'alliberaran perquè passin a formar part del nucli del programari.

1.3 Estudi situació actual

Per estudiar la situació actual, prèvia al projecte, s'haurà de procedir a analitzar tres aspectes: l'eina sobre la que es realitzarà la simulació, el BTC-NS; l'estat en el món real pel que fa a les aranyes i la xarxa distribuïda Bitcoin i, per últim, la literatura acadèmica que hagi tractat prèviament l'objecte del treball final de carrera.

1.3.1 BTC-NS: l'eina de simulació

El simulador BTC-NS és una eina de programari desenvolupada en Python que té per objectiu permetre la simulació discreta d'una xarxa d'iguals Bitcoin, sense interactuar amb la xarxa real, tant des del punt de vista de l'evolució de la xarxa distribuïda com des del punt de la tecnologia de la cadena de blocs (o *blockchain* en anglès).

El programa disposa de dos modes de funcionament: el normal i l'avançat. El primer permet definir els nodes que hi ha a la xarxa i determinar un model de xarxa, que estableixi a alt nivell les seves propietats. D'altre banda, el segon permet introduir un fitxer GML [1] que descrigui per complet l'escenari, incloent la connexions dels diferents nodes.

Pel que respecta a l'emmagatzemament el programa empra una base de dades relacional MySQL per guardar els diversos esdeveniments i informació que es va generant durant la simulació.

En relació al maquinari necessari, el simulador està dissenyat per executar-se en una única màquina (o dues, si es separa la base de dades MySQL) i, en el moment de redactar el present document, es desconeixen els requisits de maquinari mínims. La interfície d'usuari està implementada amb tecnologies web programades en Python, tot i que també es pot arribar a invocar des de línia de comandes.

Un altre aspecte important és l'estat del simulador. En el moment de la redacció d'aquestes línies l'eina està sota desenvolupament, pel que no està finalitzat. No obstant l'autor del projecte n'és un dels programadors, pel que es podrien implementar les funcionalitats necessàries.

Per últim cal assenyalar que el BTC-NS ha comptat amb el suport de la fundació Bitcoin pel seu desenvolupament.

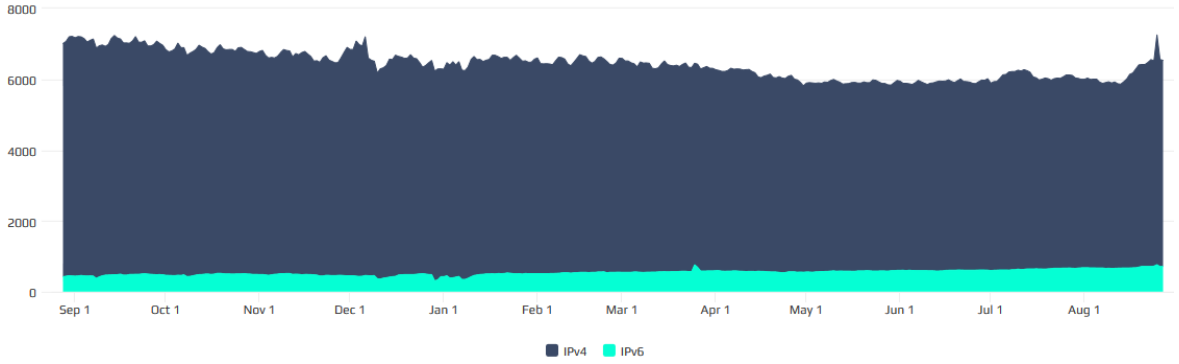
1.3.2 Aranyes en el món real

És complicat determinar la popularitat de les aranyes de Bitcoin en el món real ja que, a diferència dels altres components de la moneda criptogràfica, no tenen un component econòmic que serveixi de motivació. En aquest sentit, és possible que únicament existeixi un client públic d'aranya: Bitnodes [3].

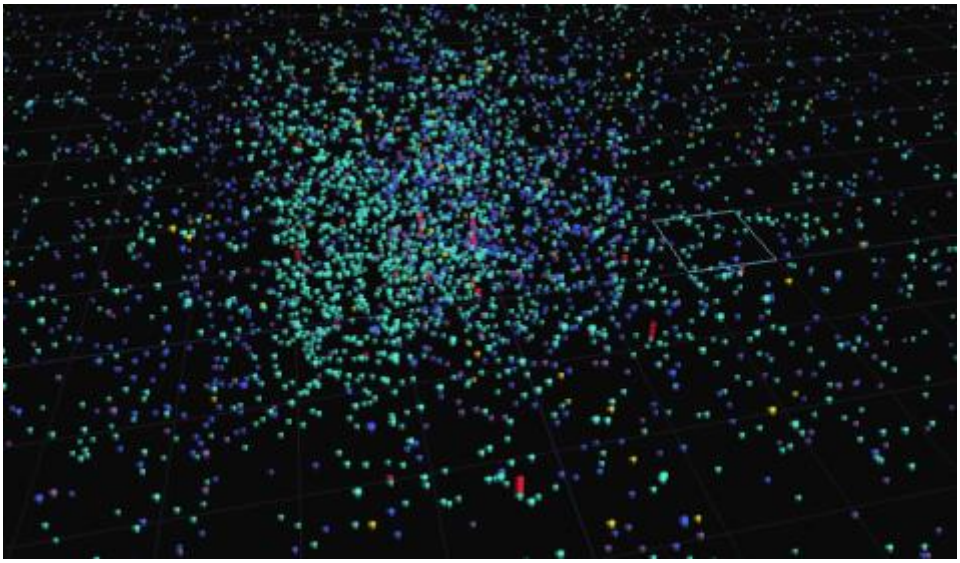
Bitnodes és un programari de codi lliure, amb una llicència personalitzada, escrit en Python per *Addy Yeow Chin Heng* amb el suport de la fundació Bitcoin. Com a aranya, el programari es dedica a escanejar la xarxa Bitcoin per obtenir els nodes que hi ha, la seva geolocalització i el seu estat (si porten molt de temps actius o s'estan sincronitzant).

Mitjançant una interfície web i una API, el programari proporciona de forma pública la informació que recol·lecta mitjançant mapes geolocalitzats i de topologia de xarxa; i taules amb informació sobre la ubicació i la versió del client (per exemple "/Satoshi:0.10.2/ (70002)" o "/Bitcoin XT:0.11.0/ (70010)") dels nodes que troba.

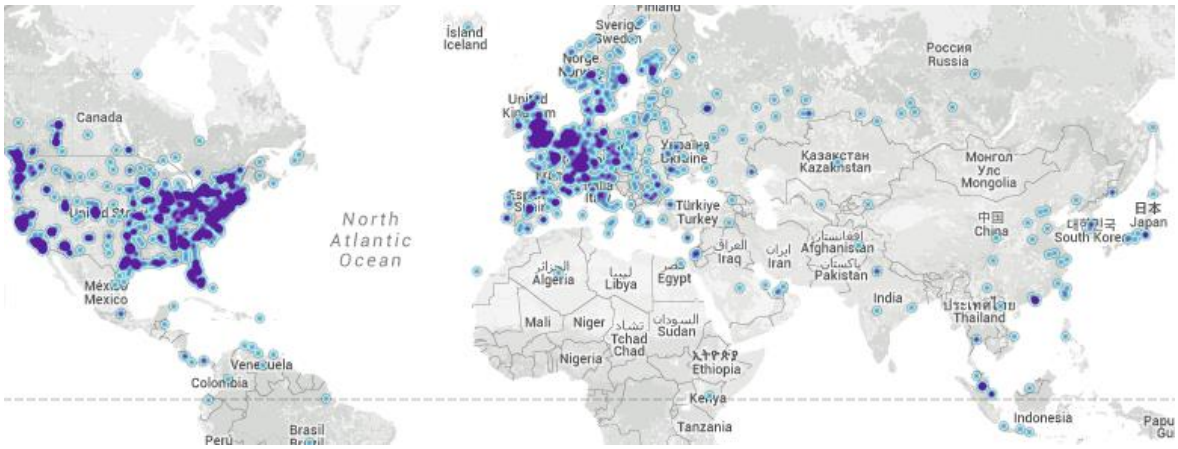
A dia d'avui, la informació que proporciona aquest programari és la millor estimació disponible sobre la mida de la xarxa Bitcoin. A tall d'exemple a 26 d'agost de 2015, d'acord a la informació recollida per https://getaddr.bitnodes.io/, la xarxa està formada per 6.686 nodes, dels quals la majoria utilitzen un client Satoshi, seguits pel client BitcoinXT.



Imatge 1: evolució del número de nodes en l'últim any. Bitnodes (2015-08-28)



Imatge 2: exemple de la topologia de xarxa obtinguda per Bitnodes (2015-08-20)



Imatge 3: exemple de la geoposició dels nodes obtinguda per Bitnodes (2015-08-20)

D'altra banda, es pot especular sobre si les empreses especialitzades disposen de les seves pròpies dades privades per al seu anàlisi. Per exemple Blockhain publica informació sobre els miners que es troben connectats amb els seus servidors [4], tot i que aquest fet està ben allunyat del comportament d'una aranya.

1.3.3 L'estudi de les aranyes per descobrir la xarxa

Les aranyes de Bitcoin, així com la tipologia de la pròpia xarxa, sembla que no és per si sol un element d'especial interès per la comunitat acadèmica. La majoria d'estudis que tracten la tipologia de la xarxa s'enfoquen a qüestions relacionades amb la seguretat i, sobretot, l'anonimat.

En aquest sentit podem trobar escrits que empren aranyes com el desenvolupat per Biryukov, Khovratovich i Pustogarov [5] que utilitza l'aranya Bitnodes per obtenir els servidors de Bitcoin actius per poder analitzar la transmissió dels diferents elements de la xarxa descentralitzada.

No obstant, si que existeixen articles que estudien la pròpia topologia de la xarxa. Exemples d'aquests articles serien:

- Donet, Pérez-Solà i Herrera-Joancomartí [6] intenten determinar la mida de la xarxa P2P Bitcoin, la seva distribució geogràfica i l'estabilitat dels nodes de la xarxa real mitjançant una aranya .
- Miller, Litton i altres [7] proposen un mecanisme denominat *AddressProbe* per analitzar la tipologia de la xarxa amb l'objectiu de descobrir els nodes més influents mitjançant l'anàlisi de les seves connexions.

Respecte a [6], el document és d'especial interès ja que detalla un conjunt de limitacions, exposades a continuació, per descobrir la tipologia de xarxa Bitcoins real que tenen rellevància pel projecte que es vol dur a terme. En concret:

A. "*The number of nodes discovered* […] *does not represent the entire network.* […] *some nodes do not respond to getaddr messages, so no information about their neighborhood can be extracted* […] *the standard implementation of the Bitcoin client does not return all the node's neighbors* […] *just the minimum between 23% of the active nodes and a constant, which is set to 2500.* […]"
B. "[...] *we are dealing with Bitcoin nodes* [… not …] *Bitcoin users. It is important to stress such distinction, because the usage of light-clients as well as online Bitcoin accounts is very extended, and thus an important part of Bitcoin users can not be identified as Bitcoin nodes.*"
C. "[...] *nodes may be running on machines with dynamic IP addresses. Therefore, nodes may appear to be more unstable than they really are.*"
D. "*Each of the scans took about 2 hours to complete. Therefore, some parts of the network may have changed while we were exploring other parts.* […]"

L'estudi que es vol dur a terme permetrà proporcionar un altre punt de vista complementari a algunes de les limitacions de l'article. En particular la simulació permetrà conèixer, d'una banda, la informació recollida per l'aranya i, de l'altra, la informació completa de la xarxa i per tant:

A. Es podrà estudiar l'habilitat d'una aranya per obtenir una imatge de la topologia de la xarxa, comparant-la amb la tipologia completa simulada, i permetia avaluar diverses algorismes en el mateix context.
B. Tot i que l'aranya no podrà obtenir el número d'usuaris de Bitcoin, per la mateixa raó exposada en l'article, sí que es podrà conèixer el seu número en la simulació.
C. Si el simulador ho permet, es podrà conèixer els canvis d'IP d'un node, així com quan passi a estar actiu o inactiu.

El següent punt a considerar és [7], que identifica més limitacions pel que fa al descobriment de la tipologia de xarxa:

A. Els nodes es poden trobar en una situació on no acceptin més connexions, pel que no respondrien als missatges d'una aranya per obtenir la llista d'adreces.

B. Els nodes poden estar situats darrere d'un sistema de traducció d'adreces (o *NAT* pel seu acrònim en anglès) o un tallafocs, pel que únicament es descobririen en el moment que fessin una connexió.
C. Pot succeir que no es puguin enviar els suficients missatges per obtenir adreces.

En un altre ordre de coses totalment diferent, per analitzar el comportament de les aranyes en la xarxa de Bitcoin, també ens podem remetre en estudis centrats en la topologia d'altres xarxes de d'igual a igual com per exemple *Gnutella*. Exemples d'aquest tipus d'articles serien Lv, Cao i altres [8] i Schlosser, Condie i Sepandar [9].

1.4 Anàlisi de la necessitat del projecte

Bitcoin al igual que qualsevol xarxa de descentralitzada d'iguals es caracteritza per la dificultat en determinar la tipologia de la xarxa. Aquest fet és clarament comprensible si assumim que, en una xarxa pública, cada node podria estar controlat per un actor diferent.

Per consegüent, per tal de conèixer la xarxa real cal desenvolupar mecanismes per descobrir tots els nodes que n'hi formin part. En aquest sentit, per obtenir una visió de la xarxa cal emprar un programa tipus "aranya" per recórrer tots els nodes disponibles i, per cada un d'ells, esbrinar quins són els seus veïns.

Tanmateix, en una xarxa informàtica d'aquestes característiques ens podem trobar amb diversos elements –per exemple tallafocs, sistemes de traducció d'adreces, nodes que no acceptin més connexions o la pròpia evolució– que impedeixin o dificultin obtenir-ne una visió acurada. Per tant, tota aranya que analitzi una xarxa tindrà un grau d'error.

Considerant que en el món real és impossible determinar el grau d'error –en el benentès que caldria conèixer tots els elements de la xarxa, el que és impossible–, cal realitzar aproximacions alternatives com, per exemple, emprar simulacions per obtenir dades que permetin extrapolar el funcionament en la realitat.

Per tant, conèixer el grau d'efectivitat d'una aranya en l'exploració de la xarxa pot contribuir, a banda del propi interès acadèmic, a facilitar la precisió d'estudis futurs al poder disposar d'estimacions de l'error de les aranyes al descobrir els diferents nodes de Bitcoin i, fins i tot, col·laborar en el disseny d'algorismes d'exploració més efectius.

A més a més, com que el simulador BTC-NS acabarà el seu desenvolupament de forma paral·lela en el projecte, l'estudi de l'aranya presentarà una bona oportunitat per tal de provar-lo realment i que els seus desenvolupadors puguin rebre retroalimentació sobre el seu funcionament.

2 Pla de treball

Pel desenvolupament del projecte s'estableix un paquet de treball per cadascun dels objectius anteriorment definits en l'apartat 1.2:

Codi	Descripció
WP1	Estudi i documentació del protocol.
WP2	Anàlisi de la xarxa Bitcoin en l'àmbit de la xarxa.
WP3	Anàlisi de la xarxa simulada en el BTC-NS en l'àmbit de la xarxa.
WP4	Desenvolupament de les millores del simulador.
WP5	Estudi de l'aranya en l'entorn simulat.
WP6	Documentació formal del treball

Taula 5: paquets del treball del projecte.

2.1 Tasques

I per cada un dels paquets de treball s'han establert les següents tasques:

WP1: estudi i documentació del protocol.	
Descripció:	Documentar el funcionament bàsic del protocol Bitcoin
Inici:	25/08/2015
Fi:	05/09/2015
Tasca 1.1:	Descriure el funcionament bàsic del protocol Bitcoin, enumerant les principals característiques.
Tasca 1.2:	Descriure el funcionament bàsic de la xarxa Bitcoin

Taula 6: tasques del paquet de treball 1

WP2: anàlisi de la xarxa Bitcoin en l'àmbit de la xarxa.	
Descripció:	Analitzar i documentar el mètode de treball "estàndard" de Bitcoin pel que respecta la xarxa d'iguals..
Inici:	06/09/2015
Fi:	06/10/2015
Tasca 2.1:	Localitzar i estudiar documentació de referència.
Tasca 2.2:	Analitzar el codi del client de referència "Bitcoin-Core".

Taula 7: tasques del paquet de treball 2

WP3: anàlisi de la xarxa simulada en el BTC-NS en l'àmbit de la xarxa.	
Descripció:	Analitzar i documentar el procediment
Inici:	06/09/2015
Fi:	06/10/2015

WP3: anàlisi de la xarxa simulada en el BTC-NS en l'àmbit de la xarxa.	
Tasca 3.1:	Estudiar i documentar el funcionament de la xarxa d'iguals del simulador.
Tasca 3.2:	Estudiar i documentar les dades que emmagatzema el simulador per determinar si són suficients per l'estudi de l'aranya.
Tasca 3.3:	Detectar els aspectes a millorar en el simulador.

Taula 8: tasques del paquet de treball 3

WP4: desenvolupament de les millores del simulador.	
Descripció:	Implementar les millores detectades en la tasca T3.2
Inici:	07/10/2015
Fi:	20/10/2015
Tasca 4.1:	Implementar en el simulador les millores detectades en la T3.2
Tasca 4.2:	Joc de proves i validació dels resultats.

Taula 9: tasques del paquet de treball 4

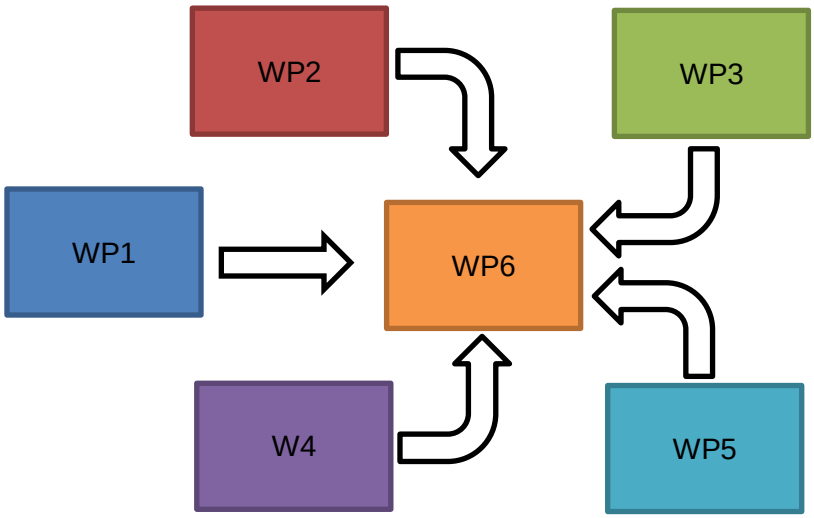
WP5: estudi de l'aranya en l'entorn simulat.	
Descripció:	Estudiar el comportament de l'aranya
Inici:	21/10/2015
Fi:	08/11/2015
Tasca 5.1:	Configuració del simulador en una màquina on executar les simulacions.
Tasca 5.2:	Determinar les simulacions necessàries a dur a terme per obtenir dades.
Tasca 5.3:	Executar les simulacions.
Tasca 5.4:	Estudi i interpretació dels resultats.

Taula 10: tasques del paquet de treball 5

WP6: documentació formal del treball	
Descripció:	Documentar formalment els treballs del projecte en una memòria.
Inici:	20/08/2015
Fi:	15/11/2015
Tasca 6.1:	Confeccionar la taula de continguts de la memòria final del projecte.
Tasca 6.2:	Documentar formalment el treball realitzat.

Taula 11: tasques del paquet de treball 6

Aquestes tasques s'interrelacionen tal com es mostra a continuació:



Imatge 4: relació entre els diferents paquets de treball

2.2 Principals fites del projecte

Les fites del projecte coincidiran amb l'entrega de les proves d'avaluació continua pel que s'estableixen les següents fites:

Data	PAC	Tasques incloses
2015-10-07	PAC1	Redacció del present document. Finalització del WP1 i WP2. Inici del WP3
2015-11-09	PAC2	Finalització del WP3. Inici del WP4 i WP5
2015-12-18	PAC3	Finalització del WP4 i WP5.
2015-01-04	Final	Finalització del WP6.

Taula 12: fites del projecte

2.3 Productes a lliurar

En cada fita s'entregaran els següents documents:

PAC	Productes a lliurar
PAC1	– Pla de treball (el present document) – Preparar la taula de continguts de la memòria del projecte – Documentar la introducció de Bitcoin. – Documentar l'estudi del funcionament de la xarxa. – Anàlisi de l'estat del projecte
PAC2	– Document amb l'estudi del funcionament del BTC-NS. – Document amb les tasques a dur a terme en l'estudi, així com la seva metodologia. – Anàlisi de l'estat del projecte

PAC	Productes a lliurar
PAC3	– Implementació de les millores en el simulador. – Document amb els resultats del projecte. – Anàlisi de l'estat del projecte
Final	– Memòria final del projecte.

Taula 13: llista de productes a lliurar en el projecte

2.4 Planificació

Per tal de poder finalitzar el projecte a desembre de 2015, les tasques s'iniciaran a mitjans d'agost de 2015. En aquest sentit com que en el projecte únicament hi intervé una persona la seva planificació temporal la plasmarem, de forma simplificada, en una única taula organitzada per mesos i setmanes:

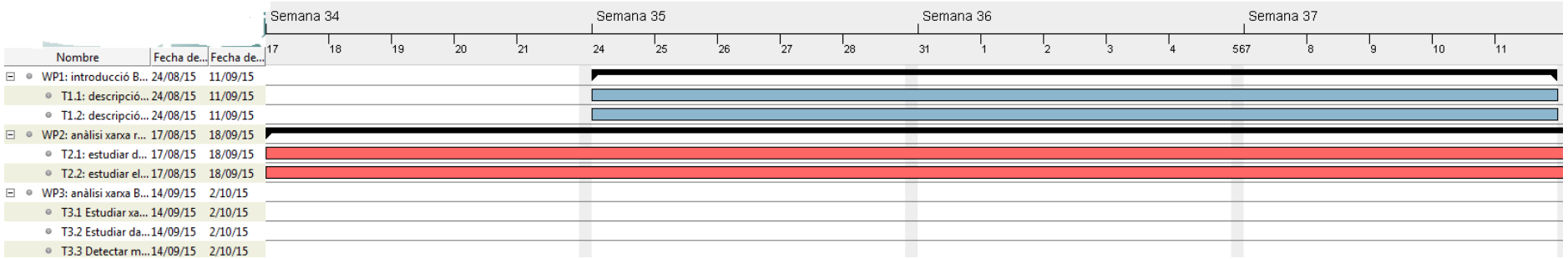
Mes	Setmana	Tasques a dur a terme
Agost	1	N/A
	2	N/A
	3	– Fi del pla de treball del projecte – WP2: inici de l'estudi del protocol de xarxa de Bitcoin. [T2.1] [T2.2]
	4	– WP1: inici de l'escriptura de la introducció a Bitcoin [T1.1] – WP2: estudi del protocol de xarxa de Bitcoin. [T2.1] [T2.2] – WP6: esborrany la taula de continguts de la memòria [T6.1]
Setembre	1	– WP1: fi de l'escriptura de la introducció a Bitcoin [T1.1] [T1.2] – WP2: estudi del protocol de xarxa de Bitcoin. [T2.2] – WP6: acabat l'esborrany de la taula de continguts [T6.1]
	2	– WP2: fi de l'estudi del protocol de xarxa de Bitcoin – WP3: inici de l'estudi del protocol de xarxa del simulador [T3.1]
	3	– WP3: estudi del protocol de xarxa del simulador [T3.1] [T3.2] [T3.3]
	4	– WP3: fi de l'estudi del protocol de xarxa del simulador. – WP4: inici de la implementació de les millores [T4.1]
Octubre	1	– WP4: implementació de les millores [T4.1] [T4.2] – Fita 1: PAC1
	2	– WP4: fi de la implementació de les millores [T4.1] [T4.2]
	3	– WP5: definir la simulacions que es duran a terme [T5.1] [T5.2] – WP5: inici de l'execució de les simulacions [T5.3] [T5.4]
	4	– WP5: estudi i interpretació dels resultats [T5.4]
Novembre	1	– WP6: redacció de la memòria. – Fita 2: PAC2
	2	
	3	Temps reservat per compensar els possibles endarreriments.
	4	
Desembre	1	
	2	
	3	- Fita 3: PAC3
	4	Temps reservat per imprevistos.
Gener 2016	1	
	2	- Fita 4: PAC4

Taula 14: planificació del projecte

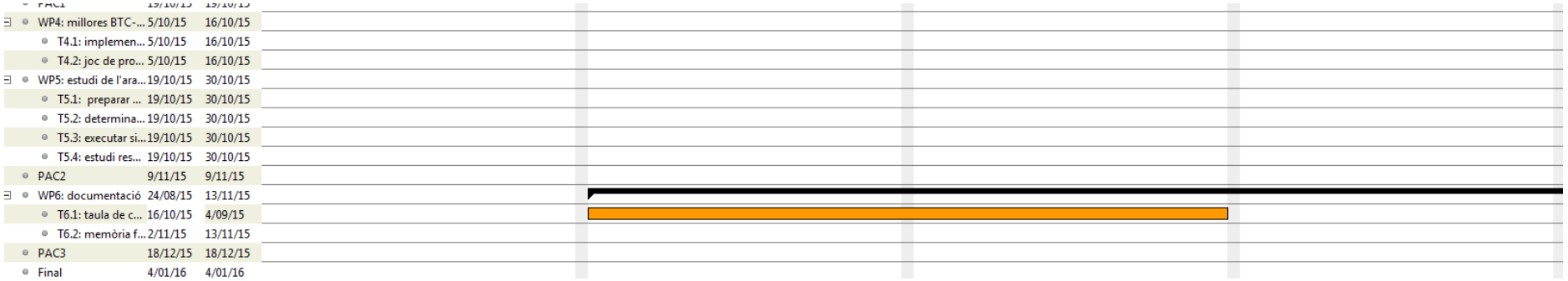
Gràficament podem representar la taula anterior mitjançant diagrames de Gantt tal com es mostra en les imatges de la següent pàgina.

2.5 Requeriments del projecte

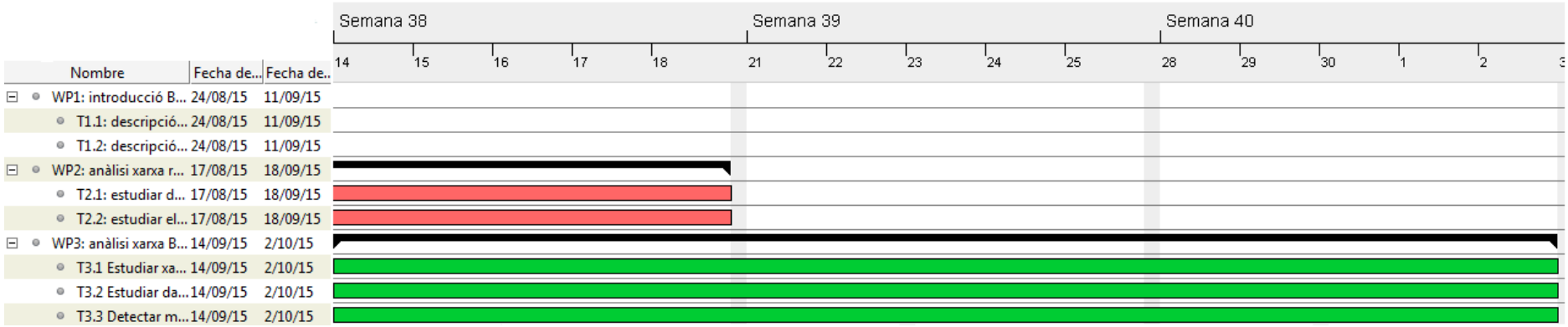
Per tal de dur a terme el projecte serà necessari disposar d'una màquina on s'executi la simulació, en aquest sentit es podria utilitzar servidors personals del projectista o de tercers, com per exemple, de la Universitat, d'Amazon o de Google. L'elecció dependrà en bona mesura de les necessitats per executar la simulació.



Imatge 5: diagrama de Gantt, setmanes 34 a 37, WP 1 a 3



Imatge 6: diagrama de Gantt, setmanes 34 a 37, WP 4 i memòria



Imatge 7: diagrama de Gantt, setmanes 38 a 40, WP 1 a 3

Imatge 8: diagrama de Gantt, setmanes 38 a gener 2016, WP4 i memòria

3 Metodologia del projecte

Pel que fa a la metodologia a seguir en el projecte es pot diferenciar en dos grans blocs: la metodologia del treball (paquets WP1, WP2, WP3, WP4 i WP6) i la metodologia de l'estudi (WP5).

3.1.1 Metodologia del treball

Pel que fa el treball no s'estableix una metodologia formal, no obstant això es seguiran les següents guies:

- Setmanalment es farà una planificació i seguiment de les tasques a implementar.
- Tota la documentació s'emmagatzemarà en serveis al núvol per tal de garantir-ne còpies de seguretat.
- El codi desenvolupat pel simulador s'introduirà en un sistema de control de versions. La política de *commits* es decidirà d'acord al tipus de repositori que s'utilitzi (subversion o git) i també dependrà de si els canvis s'introdueixen directament al nucli del simulador o es realitzà una bifurcació pel projecte.

En el capítol 7.1 s'amplia la metodologia que es seguirà per la codificació de les millores.

3.1.2 Metodologia de l'estudi

La metodologia de l'estudi es desenvolupa en llarg del capítol 8.2, un cop analitzades la xarxa de Bitcoin. No obstant això, es poden avançar diversos aspectes que s'hauran de tenir presents:

- Algunes de les simulacions hauran de disposar un alt volum de nodes, per tal d'assegurar-nos la similitud en el món real.
- Caldrà obtenir una imatge de la topologia completa de la xarxa així com de les dades obtingudes per l'aranya.
- Caldrà executar diverses simulacions per assegurar-se que s'obtinguin dades suficients.

D'altre banda, en el capítol 8.2 concretem la metodologia que es seguirà al llarg de l'estudi.

4 Introducció a Bitcoin

Al llarg d'aquest capítol proporcionarem al lector una introducció a Bitcoin per tal que es pugui situar en context. En primer lloc mostrarem una visió general del seu funcionament; tot seguit procedirem a desgranar alguns dels seus components tecnològics i, per últim, explicarem la lògica de negoci que sosté la moneda criptogràfica.

4.1 Què és Bitcoin?

D'acord a la descripció realitzada pel seu creador [10], Bitcoin és un sistema de diners electrònics que permet enviar efectiu entre dues parts sense necessitat d'una institució financera intermediària. Per aconseguir-ho es basa en dos mecanismes: criptografia de clau pública i un mecanisme de confiança implementat sobre una xarxa d'iguals o *P2P*.

Bitcoin s'engloba dintre de l'àmbit de les monedes criptogràfiques: un sistema de diners virtuals que permet operar de forma relativament segura gràcies a l'ús extensiu en els seus fonaments de la criptografia, en especial funcions resum i mecanismes de signatura amb clau pública[3].

Pel que fa al seu funcionament, a grans trets es pot explicar realitzant un paral·lelisme amb un sistema de pagarés. Els pagarés al portador són documents on consta que el seu posseidor disposa d'una certa quantitat de diners en una determinada divisa que li ha d'abonar una tercera persona.

Als pagarés s'hi poden afegir condicions addicionals, que impedeixin el seu ús fins que es complexin, i es poden vendre o transferir a tercers. Per tal d'assegurar-ne la validesa es sol acudir a una entitat intermediària –ja sigui un banc o un notari– per tal de poder garantir de forma fefaent l'existència del document o la seva transmissió.

Els pagarés permeten a Alicia generar i signar un document on declari que s'ha de pagar al seu receptor, Bob, una quantitat determinada. Un cop generat el document, Bob reclamarà atorgar-li una certa validesa, i s'adreçaran al notari de la seva localitat per tal d'aconseguir que garanteixi l'existència i validesa del document.

No obstant, tal com hem comentat en el primer paràgraf, Bitcoin no disposa d'una institució intermediària, pel que la figura del notari no existeix, i cal trobar un altre mecanisme que permeti assegurar-ne la validesa. Per aconseguir-ho, Alicia i Bob opten per emetre'n diverses còpies originals i repartir-les entre tots els seus veïns.

Amb aquest mètode s'aconsegueix deixar de dependre d'una autoritat central –el notari– emprant una "autoritat" col·legiada i descentralitzada: un nombre indeterminat de veïns. Per tant, si una tercera part vol comprovar l'existència d'un document en lloc d'adreçar-se a una persona en concret haurà d'adreçar-se a N veïns per comprovar si en disposen d'una còpia vàlida.

Conseqüentment, Alicia, en lloc de donar-li el pagaré únicament a Bob, en repartirà còpies entre tots els seus veïns (incloent a Bob). En cas que Bob no l'arribés a rebre, pel motiu que sigui, podrà preguntar a les persones pròximes per comprovar si el tenen i, en cas afirmatiu, que li entreguin una còpia.

D'altra banda, Bob també voldrà determinar si realment Alicia té els diners necessaris per generar el pagaré i, per tant, perquè pugui disposar dels diners. Per sort, en tots els

[3] Per ampliar més informació el lector pot consultar la descripció oferta per Wikipedia: https://en.wikipedia.org/wiki/Public-key_cryptography.

documents hi figura d'on provenen els fons i conseqüentment es pot establir una cadena i conèixer si Alicia ha rebut realment els diners necessaris per poder fer-hi front[4].

A la vegada, Bob també voldrà assegurar que Alicia no estigui emeten dos pagarés utilitzant els mateixos fons. Però com en aquests documents hi figura l'origen dels diners, és a dir, quin pagaré previ provenen, Bob pot consultar-los mitjançant les còpies que posseeix i concloure si Alicia esta intentant gastar-se dos cops els mateixos diners.

Tot això, al mateix temps, comporta que el volum de treball corresponent a la classificació, organització i certificació dels pagarés que anteriorment realitzava una autoritat central –el notari– a canvi d'una remuneració, ara recau sobre N individus que ho realitzen de forma altruista i totalment voluntària.

Aquest fet implica que a mesura que el volum de feina augmenti, alguns dels veïns deixaran de realitzar aquesta tasca no remunerada. Per evitar-ho cal proporcionar algun tipus d'incentiu que els motivi i compensi el seu temps ja sigui, per exemple, una propina cada cop que reben un pagaré o recompenses per agrupar i enquadernar els documents.

Si continuem amb el pagaré d'en Bob, un cop hagi realitzat totes les comprovacions anteriors, podrà validar, acceptar el pagaré d'Alicia i considerar que disposa dels fons. Quan necessiti utilitzar-lo, podrà emetre un nou pagaré, identificant el document d'on provenen els fons, i signar-lo per demostrar que n'és l'emissor.

De la metàfora anterior, que intenta explicar el funcionament de Bitcoin, se'n poden extreure les seves principals característiques:

- La descentralització, mitjançant un sistema comptable públic (la cadena de blocs o *BlockChain*) i distribuït (sobre una xarxa d'iguals P2P).
- La confiança, combinant la cadena de blocs, la xarxa descentralitzada d'iguals i un sistema d'incentius (recompenses per dur a terme tasques necessàries pel funcionament de Bitcoin) que pot arribar a solucionar el problema dels generals bizantins[5].
- Evita el problema de la doble despesa gràcies a un sistema comptable públic, descentralitzat i d'iguals.
- Inexistència de comptes bancaris i balanços: no existeix un sistema de comptes bancaris associats a l'usuari, en el seu lloc l'usuari disposa de *drets* per disposar d'una quantitat determinada de Bitcoins. Seran aquests *drets* el que es transferiran entre usuaris.

A més a més, Bitcoin també disposa d'un cert anonimat, ja que en les transaccions no es pot identificar fàcilment l'emissor o el receptor ja que s'utilitza un sistema de pseudònims. Aquest fet és d'especial rellevància si es considera que el sistema comptable de Bitcoin és totalment públic.

4.2 L'ús de Bitcoin

En termes més pràctics, per utilitzar Bitcoins únicament cal descarregar-se un dels diversos moneders existents[6] (ja sigui un programa a instal·lar en un PC o telèfon mòbil, o bé un servei web) i adreçar-se a una casa de canvi[7], serveis que permeten obtenir moneda virtual mitjançant un tipus de canvi establert d'acord al mercat.

[4] Suposant que treballem en un sistema on únicament s'empren pagarés.
[5] Tot i que alguns autors, com Grigg [25], sostenen que no soluciona el problema, l'esquiva.
[6] Una llista es pot trobar a https://bitcoin.org/es/elige-tu-monedero [data de consulta 2015-01-02].
[7] Una llista es pot trobar a http://howtobuybitcoins.info/#!/EUR [data de consulta 2015-01-02].

Un moneder típic de Bitcoin permet, com a mínim: realitzar transaccions, és a dir enviar Bitcoins entre dos usuaris; generar peticions de pagament (un sistema que facilita a l'emissor l'adreça on han de remetre una quantitat determinada de Bitcoins) i generar adreces, el pseudònim on un usuari podrà enviar o rebre Bitcoins.

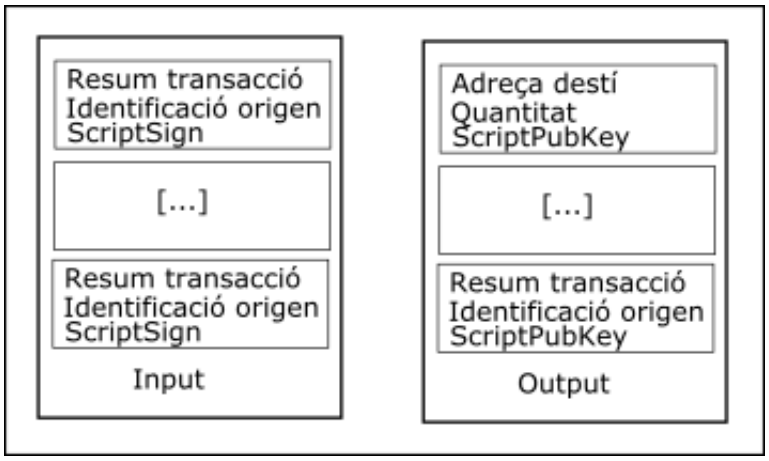
Amb les funcionalitats d'un moneder l'usuari ja pot generar adreces, adreçar-se a una casa de canvi per obtenir Bitcoins a preu de mercat, rebre'ls a les adreces que s'han generat en el primer pas i, finalment, realitzar transferències enviant els Bitcoins a adreces de tercers. En aquest punt ja podria utilitzar Bitcoins.

4.3 Components tecnològics de Bitcoin

Des de la perspectiva tecnològica, tal com veurem a continuació, el funcionament de Bitcoin es basa en transaccions, que és el mecanisme pel qual els usuaris es transmeten Bitcoins; la cadena de blocs, l'estructura comptable pública on figuren les transaccions que s'han realitzat; i, per últim, la xarxa P2P descentralitzada, que és el mitja emprat per transmetre i distribuir tota la informació.

4.4 Les transaccions

La transferència de Bitcoins entre dues o més adreces es documenta en una estructura de dades denominada *transacció*. En aquest constructe hi figuren, d'una banda, una referència a les transaccions prèvies des d'on provenen les monedes que es transferiran i, d'altra banda, les adreces on aniran a parar els diners.



Imatge 9: estructura de dades d'una transacció

Si observem la imatge anterior, podem observar com cada transacció pot tenir N entrades i M sortides, on N i M poden ser xifres diferents. Cada una de les entrades apunta a la transacció prèvia des d'on s'obtenen els Bitcoins i, de l'altra banda, cada una de les sortides identifica l'adreça a la que s'envien els diners i la seva quantitat.

Si ens hi fixem, podem observar que existeixen dos camps denominats *ScriptSig*, o *locking* script; i *ScriptPubKey*, o *unlocking* script. Aquest camps corresponen, respectivament, a la condició necessària per gastar el Bitcoin de la transacció anterior i el repte necessari per tal que una transacció posterior pugui gastar les monedes.

És a dir, Bitcoin no és un sistema que es limita a transmetre diners entre dos actors. És un sistema implementat sobre un llenguatge de programació[8] que fa púbica una condició

[8] Un llenguatge que no implementa una màquina de Turing: no permet recursivitat o iteracions i limita la longitud màxima del programa i, per tant, el seu temps d'execució.

(generalment criptogràfica). Qualsevol actor que sigui capaç de complir-la podrà assolir el control de les monedes de la transacció i gastar-les[9].

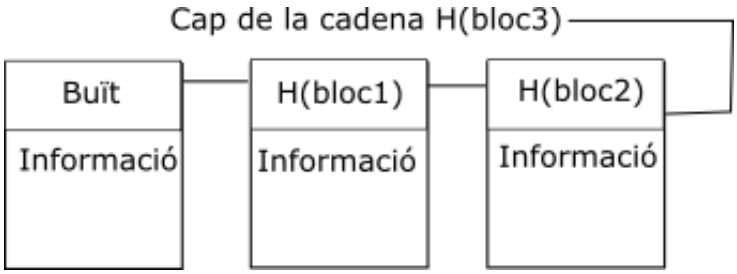
A la pràctica el 99% [11] de les condicions corresponen al mecanisme *Pay to Public Key Hash* (P2PKH). En aquest sistema *ScriptPubKey* correspon al resum de la clau pública (on s'enviaran els Bitcoins) i *ScriptSig* a la signatura digital (amb clau privada) que permet assegurar que la transacció l'ha generat el propietari de les monedes.

Aquest sistema està basat en una tecnologia denominada *contractes intel·ligents*, que queda fora de l'abast del document. Si el lector desitja aprofundir sobre aquest aspecte disposa d'una amplia bibliografia relacionada amb Bitcoin (per exemple [12] o [13]) i amb els contractes intel·ligents en general (per exemple [14] o [15]).

4.4.1 La cadena de blocs

Un altre component tecnològic de rellevància en Bitcoin és la cadena de blocs o *Blockchain*. Seguint la metàfora de l'apartat 4 la cadena de blocs seria la informació de la que disposen els veïns i on hi figuren totes les dades de les transaccions realitzades per Bitcoin des del seu inici.

Des d'una perspectiva tècnica la cadena de blocs és un sistema de classificació d'informació, que l'agrupa en unitats denominades blocs. Cada element de la cadena excepte el primer té una referència al bloc anterior, implementada com el resultat d'una funció resum. A més a més, també existeix un punter a l'últim bloc generat: el cap de la cadena.



Imatge 10: estructura de la cadena de blocs

L'estructura presenta unes interessants característiques de seguretat, ja que, en cas que un atacant decideixi alterar un bloc, no només haurà de modificar el bloc atacat, sinó que també haurà de modificar tots els posteriors més el cap de la cadena. Amb aquest mètode es dificulta, i amb suficients blocs s'impossibilita, l'alteració sense que es detecti la pèrdua de la integritat.

En Bitcoin, cada un dels blocs de la cadena conté un número determinat de transaccions, limitat per la mida màxima del bloc. Pel que fa a la seva creació, els blocs es generen per un procés de mineria[10] que, a grans trets, consisteix a afegir al bloc un número aleatori per tal que el seu resum compleixi una condició determinada consensuada per la xarxa.

[9] Pel que ens trobem en una situació de concurrència: dos actors podrien gastar a la vegada la mateixa moneda.

[10] Per ampliar informació el lector es pot adreçar a [12], que està disponible gratuïtament en http://chimera.labs.oreilly.com/books/1234000001802/ch02.html#_adding_the_transaction_to_the_ledger [data de consulta 2015-09-09]

4.4.2 Regles de negoci: incentius, protocol i consens.

Un dels altres conceptes que permeten el funcionament de Bitcoin són les seves *regles de negoci*. Amb aquest terme ens referim a aquells acords consensuats, tal com veurem més endavant, que permeten que la moneda criptogràfica funcioni. Aquestes *regles* inclouen els incentius, el protocol de Bitcoin i el propi consens.

Pel que fa el primer, els incentius, tal com hem vist en l'apartat 4.1 cal proporcionar algun tipus de motivació als participants per tal que realitzin voluntàriament tasques imprescindibles pel funcionament de la moneda criptogràfica. Aquesta motivació sorgeix en el procés de mineria, on s'atorguen Bitcoins per validar transaccions i agrupar-les en blocs.

En segon lloc, s'ha d'establir el seu protocol de funcionament, per tal de conèixer com s'han de comunicar els nodes de la xarxa i el format de les estructures de dades que intercanvien (per exemple la mida màxima dels blocs). Al no existir una autoritat central que reguli el seu funcionament es va optar per una regulació consensuada[11].

Per últim, el consens es realitza per *votació* de la majoria dels nodes de la xarxa. En aquest sentit el protocol que utilitza Bitcoin serà el que emprin la majoria dels seus participants. Conseqüentment, si un únic actor aconsegueix controlar més del 50% de la xarxa, podria obtenir un gran domini sobre Bitcoin.

En aquest punt cal diferenciar el concepte de *node* o *participant* de la xarxa del d'*usuari* de Bitcoin. El primer és un participant actiu de la xarxa (genera, retransmet o valida transaccions o blocs), mentre que el segon utilitza un *node* per tal d'operar amb els seus Bitcoins. Per tant, un node pot agrupar un nombre indeterminat d'usuaris.

4.5 La xarxa Bitcoin

En l'exemple que hem utilitzat en la introducció del capítol, exposàvem que tota la informació de les transaccions es distribuïa entre els veïns d'Alicia i Bob. Al igual que en l'exemple, en Bitcoin també cal distribuir la informació del sistema entre tots els membres de la xarxa.

Publicar informació a Internet és un mecanisme relativament senzill, que es pot implementar, per exemple, mitjançant un servidor web. No obstant, publicar dades de forma descentralitzada, és una mica més compleix i Bitcoin ho adreça distribuint la informació continguda en la cadena de blocs entre tots els participants d'una xarxa d'iguals.

D'aquesta forma, cada participant de la xarxa pot obtenir una còpia completa de la cadena de blocs, el que li permet disposar de tota la informació comptable de Bitcoin des dels seus inicis. Conseqüentment qualsevol node pot arribar verificar si les transaccions que rep tenen fons i que no existeixi una doble despesa.

La integritat de la informació en aquest sistema distribuït s'aconsegueix mitjançant una combinació de funcions resum, per assegurar que no s'alterin els blocs i conèixer quin és el bloc previ, criptografia de clau pública, per assegurar tant la integritat de les transaccions com qui n'és l'autor, i les regles de negoci.

Pel que respecta al funcionament de la xarxa, i tal com detallem en el següent capítol, aquesta treballa en Internet sobre els protocols IP (tant la versió 4 com la 6) i TCP. I, a

[11] És convenient aclarir que no existeix una definició formal del protocol Bitcoin.

nivell d'aplicació, treballa damunt d'un protocol de comunicacions propi de Bitcoin. Addicionalment, Bitcoin també permet treballar sobre altres protocols com la xarxa TOR[12].

El protocol de Bitcoin implementa totes les funcionalitats necessàries per establir la connexió a nivell d'aplicació i permet la comunicació mitjançant un sistema de missatgeria. Són aquests missatges els que permeten la tramesa de dades entre els iguals (en l'annex I es proporciona una llista amb els missatges existents).

Per últim, per damunt de la xarxa de Bitcoin trobem els nodes que, tal com hem indicat abans, són qui realitzen les operacions sobre la xarxa. En aquest sentit cal tenir present que no tots els nodes realitzen les mateixes operacions, pel que els podem arribar a classificar tal com es mostra en l'annex II.

[12] El lector pot trobar més informació sobre TOR a la pàgina web del projecte: https://www.torproject.org/ [data d'accés 2015-09-28].

5 La xarxa Bitcoin

Per tal de disposar del coneixement necessari per mimetitzar amb fidelitat el comportament del mecanisme de descobriment de veïns de la xarxa d'iguals Bitcoin en el simulador BTC-NS, en aquest capítol n'analitzarem el seu funcionament, centrant-nos en el mecanisme per trobar els iguals de la xarxa.

Per aconseguir documentar el seu funcionament ens hem d'enfrontar primer de tot a dos qüestions: d'una banda, no existeix una definició formal del protocol de comunicació de Bitcoin i, d'altra banda, al igual que qualsevol xarxa d'iguals, cada participant de la xarxa pot usar una solució de programari que implementi algorismes diferents.

Per tant, abans de començar l'anàlisi cal determinar quina aproximació es segueix per afrontar aquests dos inconvenients. En aquest sentit cal considerar que, tot i que teòricament el funcionament de Bitcoin es determina per consens, hi ha un programari que té una gran influència sobre la xarxa: Bitcoin Core [16].

Bitcoin Core, que també es coneix com "el client de referència de Bitcoin", és un programari desenvolupat inicialment pel creador de la moneda criptogràfica –Satoshi Nakamoto– i es considera que implementa el funcionament *estàndard* de Bitcoin, tant pel que fa les comunicacions com altres elements de la tecnologia.

Conseqüentment, si es procedeix a analitzar el seu funcionament, es pot arribar a disposar d'una bona visió de com treballa la xarxa d'iguals que sosté a Bitcoin. No obstant, tal com hem indicat abans, l'estudi del funcionament del client no es pot extrapolar a com treballen tots els participants de Bitcoin.

Sigui com sigui, l'anàlisi de Bitcoin Core haurà d'incloure els següents aspectes:

- Mètode de transmissió de dades de connexió entre els nodes.
- Mètode de gestió de les connexions.
- Descobriment de nodes (diferenciant el descobriment inicial del que es realitza habitualment).
- Estructures de dades per l'emmagatzemament de les adreces dels nodes.
- Gestió, emmagatzemament i selecció d'adreces.
- Estructures de dades per emmagatzemar dades relatives als nodes.

Per últim, per concloure l'apartat plantejarem les limitacions de l'estudi:

- L'anàlisi del programa es centrarà en el funcionament amb adreces IP sense diferenciar si corresponen a la quarta o sisena versió del protocol[13]. Per tant no es consideraran altres tipus d'adreces com les de TOR.
- El codi analitzat s'ha obtingut del projecte *bitcoin* allotjat a GitHub [16], en concret del commit amb identificador 229fb97 de la rama principal del projecte (obtinguda a principis de setembre de 2015).
- L'estudi es centra en el mètode de descobriment de parells, pel que no documentarem qüestions que s'allunyin en excés d'aquesta funcionalitat (com, per exemple, l'emmagatzemament de les adreces en base de dades o la prohibició d'interactuar amb determinats nodes de la xarxa[14]).

[13] Bitcoin Core emmagatzema totes les adreces en format IPv6. Les adreces IPv4 les guarda en la sisena versió mitjançant la seva integració en una IPv6 (::FFFF:0:0/96).
[14] *Banneig* de nodes.

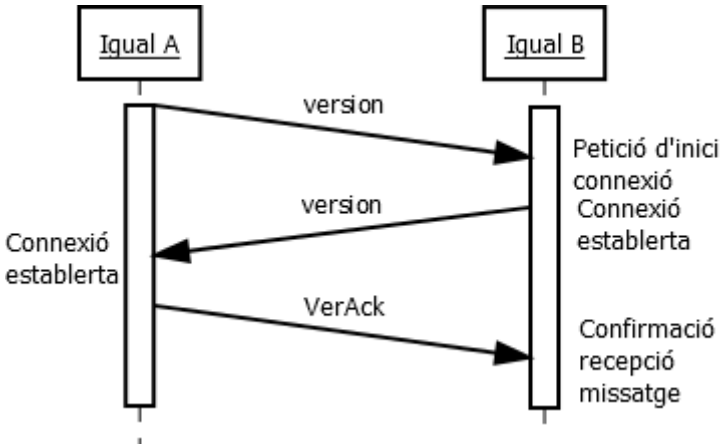
- Tampoc detallarem els mecanismes de codificació dels missatges. És a dir, no explicarem com els algorismes d'alt nivell codificaran la informació en format binari per tal de comunicar-la per la xarxa.

Al llarg d'aquest capítol veurem com es realitza la connexió de xarxa (apartat 5.1), la transmissió d'adreces entre els nodes mitjançant un sistema de missatgeria (apartat 5.2), coneixerem com Bitcoin descobreix els seus veïns (apartat 5.3), analitzarem l'emmagatzemament i la gestió de les adreces (apartats 5.4 i 5.5) i, per últim, veurem la gestió de les connexions (apartat 5.6).

5.1 La connexió a la xarxa Bitcoin

Tal com hem exposat en el capítol anterior, Bitcoin es sosté mitjançant els protocols de comunicació IP i TCP i, sobre aquesta tecnologia, implementa un protocol propi que és l'encarregat de realitzar la connexió punt a punt entre dos iguals al nivell d'aplicació. En aquest sentit, la comunicació entre dos nodes només és possible quan estan connectats.

L'establiment de la connexió es realitza mitjançant un intercanvi de missatges (o *handshake*) que proporciona als dos iguals implicats la informació necessària per treballar amb l'altre extrem. En concret es dóna a conèixer la versió del protocol Bitcoin utilitzada[15], l'adreça IP, la relació de serveis oferts[16] i la disponibilitat de l'altre extrem a intercanviar informació[17].



Imatge 11: intercanvi de missatges en l'establiment de la connexió Bitcoin.

Com podem veure en la imatge anterior, el procés d'intercanvi de missatges és relativament simple. Suposant que el node A vol establir una connexió amb el node B, i un cop realitzada la connexió TCP, els passos són:

1) *A* envia un missatge *version* a *B*. Amb aquest missatge l'igual informa a l'altre part que vol establir una connexió i li proporciona la informació necessària.

[15] Bitcoin, com tota tecnologia, evoluciona ràpidament al llarg del temps i, per aquest motiu, s'ha establert un número de versió que determina la compatibilitat entre dos clients. Com a norma general dos clients que es connectin empraran la versió més petita del protocol (sempre i quan ambdues versions siguin compatibles).

[16] Que, d'acord a l'establert en el fitxer *protocol.h* de Bitcoin Core, són: xarxa (pot comunicar blocs), *getutxo* (pot enviar les transaccions pendents de gastar) i *bloom* (per clients que no disposin d'una còpia completa de la cadena de blocs).

[17] En el benentès que un igual que no completi el procés de connexió no atendrà les peticions de l'altre extrem.

2) Si *B* vol connectar-se amb *A*, li contestarà amb un segon missatge *version* (que, a la vegada, li proporcionarà tota la informació necessària per dur a terme la connexió).

   En el cas que *B* no es vulgui connectar amb *A*, podrà respondre al primer missatge *version* amb un missatge *reject* (que indica que l'últim missatge no s'ha acceptat), no contestar o tancar la connexió TCP.

3) Per últim, *A* remetrà a *B* un missatge *VerAck* (o *version acknowledgement*) indicant que ha rebut el segon missatge *version*. La remissió del missatge *VerAck* no és obligatòria (entenent-se que, tot i que la majoria dels clients l'envien, es considera establerta la connexió amb el segon missatge *version*).

Si algun dels missatges no arriba a l'altre extrem (ja sigui perquè un dels iguals no contesta o bé si el missatge s'ha perdut en la xarxa), quan transcorri un temps establert l'igual entendrà que l'altre node no està connectat i procedirà en conseqüència, tancant la connexió TCP.

No obstant això, abans de tot, per tal de realitzar una connexió cada participant de la xarxa Bitcoin ha de conèixer les adreces d'altres nodes. Tal com veurem a continuació, aquestes adreces es podran obtenir d'altres membres de la xarxa per diversos mitjans en l'anomenat "descobriment inicial de veïns".

5.2 Transmissió d'adreces entre nodes

Per tal que els participants de la xarxa puguin comunicar-se les adreces, Bitcoin implementa en el seu sistema de missatgeria dos elements, *getaddr* i *addr*, que permeten intercanviar la informació. El primer, get*addr*, és un missatge que sol·licita a un igual informació sobre les adreces de nodes coneguts. Com a resposta a aquest missatge, el client que el rebi contestarà amb un *addr,* que contindrà les adreces obtingudes de la seva agenda mitjançant la selecció múltiple d'adreces[18].

En relació a aquests missatges, tot seguit analitzem quan un node executant Bitcoin Core els envia. D'altra banda, en el annexos podem trobar més informació sobre els tipus de missatges disponibles (annex I) i el funcionament concret del sistema de missatges de Bitcoin Core (annex III).

5.3 Descobriment de veïns

El següent punt, després de veure com els diferents clients de Bitcoin s'intercanvien informació sobre les direccions de xarxa, serà determinar què realitzen els iguals amb aquest coneixement. El comportament en concret variarà depenent del programari i del tipus de node, però a grans trets s'emprarà pel descobriment de veïns.

Una de les característiques principals de les xarxes P2P descentralitzades és el desconeixement sobre qui forma part de la xarxa i, com que no hi ha una autoritat central, tampoc existeix una llista amb tots els nodes actius. Conseqüentment, qualsevol node que hi vulgui treballar abans ha de descobrir amb qui pot fer-ho: els seus "veïns"[19].

El procés per descobrir els veïns és una qüestió que comporta dos vessants: el descobriment inicial, que es realitza quan el node no coneix ningú en la xarxa (també

[18] Funció *CAddrMan.GetAddr_* que veurem en l'apartat 5.5.4.2.

[19] El terme veí implica proximitat. Al treballar en Internet la proximitat es traduirà en la possibilitat de comunicació, no en la proximitat física. És a dir, considerem com a veí els nodes amb els que ens podem comunicar.

conegut com a *bootstrap*); i el descobriment ordinari de veïns que es realitza a mesura que el programa treballa en la xarxa.

5.3.1 El descobriment inicial de veïns

Tal com comentàvem, en la seva primera execució un node no coneixerà cap altre membre de la xarxa i, per tant, no podrà treballar. En conseqüència, cal proporcionar manualment informació sobre altres nodes o acudir a uns "punts fixos" on es pugui obtenir la informació necessària sobre la xarxa.

En aquest sentit, el programari Bitcoin Core implementa els següents mecanismes per tal de conèixer els veïns inicials:

1) Introducció de nodes coneguts de forma manual per part de l'usuari.
2) Introducció de servidors DNS específics per part de l'usuari.
3) Obtenir adreces de nodes mitjançant servidors DNS "de confiança"[20] codificats en el programa.
4) Adreces " de confiança" codificades en el propi programa[21].

Les dues primeres opcions permeten a l'usuari introduir, manualment, adreces que conegui d'altres iguals de la xarxa Bitcoin o de servidors DNS que, tot i que no treballen sobre el protocol de Bitcoin, permeten obtenir altres nodes. En concret, aquests servidors de noms[22] davant d'una consulta tornen una llista de nodes Bitcoin com a registres "A".

Type	Domain Name	IP Address	TTL
A	seed.bitcoin.sipa.be	216.144.243.18	1 min
A	seed.bitcoin.sipa.be	76.105.111.24	1 min
A	seed.bitcoin.sipa.be	50.46.154.36	1 min
A	seed.bitcoin.sipa.be	104.167.111.69	1 min
A	seed.bitcoin.sipa.be	76.117.161.69	1 min
A	seed.bitcoin.sipa.be	93.185.101.76	1 min
A	seed.bitcoin.sipa.be	78.153.4.77	1 min
A	seed.bitcoin.sipa.be	59.127.15.92	1 min
A	seed.bitcoin.sipa.be	24.19.7.96	1 min
A	seed.bitcoin.sipa.be	71.255.255.105	1 min
A	seed.bitcoin.sipa.be	45.56.91.110	1 min
A	seed.bitcoin.sipa.be	64.231.124.123	1 min
A	seed.bitcoin.sipa.be	198.50.184.123	1 min
A	seed.bitcoin.sipa.be	50.136.223.128	1 min

Imatge 12: resultat d'una consulta DNS a seed.bitcoin.sipa.be.

Les dues darreres opcions, permeten al programari intentar descobrir nodes si l'usuari no li proporciona la informació necessària. De les dues opcions, la llista dels nodes DNS inclosos en Bitcoin Core es pot trobar a *src/chainparams.cpp* i, pel que respecta a les adreces codificades en el propi programa, es poden trobar en el fitxer *src/chainparams.h*.

[20] Aquests nodes DNS són mantinguts per membres reconeguts de la comunitat de Bitcoin.
[21] Nodes de "prestigi" en la xarxa, ja sigui perquè els mantenen membres reconeguts de la comunitat, funcionen correctament amb una alta disponibilitat, són veterans o la combinació de totes les opcions anteriors.
[22] El lector pot trobar abundant informació sobre DNS a Internet, un bon punt per començar és https://es.wikipedia.org/wiki/Domain_Name_System [data de consulta 2015-10-17].

```
static SeedSpec6 pnSeed6_main[] = {
  {{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0x01,0x22,0xa8,0x80}, 8333},
```

Codi 1: exemple d'adreça IPv6 i port codificat en Bitcoin Core.

Els quatre mètodes anteriors s'intenten de forma consecutiva i no s'executen si qualsevol dels anteriors ha funcionat correctament. És a dir, no s'utilitzaran els nodes codificats en el programa si funcionen els DNS implementats i, de la mateixa forma, els servidors DNS codificats no funcionaran si l'usuari introdueix uns nodes o DNS vàlids[23].

Per concloure, exposarem que el mecanisme de descobriment inicial de nodes, únicament s'executarà el primer cop que s'invoqui el programa (o bé si, per qualsevol motiu, el node es queda sense adreces a les que connectar-se). Un cop el node coneix com a mínim un altre igual, el descobriment de veïns serà l'ordinari que detallarem a continuació[24].

5.3.2 El descobriment de veïns "ordinari"

Tal com acabem d'exposar, el descobriment de veïns ordinari és el que es realitza mentre el programari funciona i consisteix en augmentar el coneixement del node de la resta de la xarxa. Per tant, caldrà que el programari recopili la informació necessària de la xarxa per optimitzar el seu funcionament[25].

En aquest sentit, Bitcoin Core implementa dos mecanismes per la recopilació d'adreces: anunciar-se a la xarxa per tal de donar a conèixer la seva existència i el reenviament de les direccions a altres nodes cada cop que un client rebi un missatge *addr*.

En primer lloc, els nodes de la xarxa **s'anuncien periòdicament** a tots els nodes amb els que mantenen una connexió. En concret el node posa la seva pròpia direcció a la cua d'adreces que s'enviarà a l'altra banda de la connexió. La periodicitat es determina en la condició implementada en *AdvertizeLocal* del fitxer *src/net.cpp*.

```
if (!IsInitialBlockDownload() && (GetTime() - nLastRebroadcast > 24 * 60 * 60))
```

Codi 2: condició que determina la periodicitat de l'anunci propi de l'adreça de xarxa

El codi anterior determina la periodicitat en que un client anuncia la seva pròpia adreça als nodes als que està connectat. És determina el moment de l'anunci quan el nombre de segons[26] transcorreguts des de l'últim anunci (controlat per la variable *nLastRebroadcast*) supera el valor 86.400 (valor corresponent als segons d'un dia).

Addicionalment, els nodes **s'anuncien al rebre una connexió d'entrada**. En l'intercanvi de missatges *version* (d'acord a l'establert en la funció *ProcessMessage* del fitxer *src/main.cpp*), el node anuncia la seva adreça local (seguint un procediment similar al

[23] Perquè un node o DNS sigui vàlid, s'ha de poder establir una connexió a nivell d'aplicació.
[24] En Bitcoin Core les adreces conegudes s'emmagatzemen en una base de dades en suport físic, pel que no caldrà executar el descobriment de veïns cada cop que s'iniciï la seva execució.
[25] Cal considerar que per un node convencional el descobriment de veïns és un mecanisme per a un fi concret (el funcionament de la moneda criptogràfica) i, per tant, no cal que acumuli totes les direccions possibles, haurà de preservar les que li siguin més convenients.
[26] El temps s'expressa en format POSIX, és a dir, amb el número de segons des de l'1 de gener de 1970.

detallat anteriorment) i sol·licita amb un missatge *getaddr* les adreces de la seva nova connexió.

Per últim, el client cada cop que rebi un missatge *addr* amb adreces, **reenviarà** cada direcció rebuda a un o dos nodes concrets[27] (que seran els mateixos durant 24 hores). Per tal que es realitzi la retransmissió cal complir dues condicions: que el node on s'enviïn les adreces utilitzi una versió del protocol adequada i que compleixi el següent:

```
if (addr.nTime > nSince &&
!pfrom->fGetAddr && vAddr.size() <= 10 && addr.IsRoutable())
```

Codi 3: condició temporal de Bitcoin Core per reenviar adreces.

És a dir:

- L'adreça no sigui més antiga de 10 minuts.
- Si no tenim pendent de processar un missatge *GetAddr* del node on reenviarem les adreces.
- Si el missatge que volem reenviar té com a molt 10 adreces.
- Si ens podem comunicar amb l'adreça que volem reenviar.

Si es compleixen les condicions, el client afegirà l'adreça a la taula *VAddrToSend*, que emmagatzema totes les adreces que s'han d'enviar al un node determinat.

5.4 Estructures de dades de les adreces dels nodes

Ara que ja em vist el comportament del programari Bitcoin Core pel que fa al descobriment de veïns, procedirem a analitzar els algorismes que implementen dites funcionalitats. No obstant, per entendre-les haurem de començar per l'aspecte més bàsic: les estructures de dades que implementen les adreces dels nodes. En concret el programari utilitza les següents estructures de dades per treballar amb les adreces IP i TCP d'altres participants de la xarxa:

Classe	Classe pare	Fitxer[28]	Funció de la classe
CNetAddr	-	src/netbase.	Representa una adreça IP
CService	CNetAddr		Representa adreça IP i port TCP.
CAddress	CService	src/protocol.	Informació addicional sobre l'igual.
CAddrInfo	CAddress	src/addrman.	Estadístiques sobre l'igual

Taula 15: estructures de dades de Bitcoin Core d'adreces IP i TCP.

La primera classe, *CNetAddr*, representa una adreça i proporciona diverses funcionalitats per treballar sobre ella: determinar a quin protocol correspon (IPv4, IPv6, TOR o altres), a quin tipus de xarxa pertany (adreça pública, adreça privada, local, túnels IPv6, ...), conèixer si es tracta d'adreça vàlida i per últim funcions de codificació.

[27] Una adreça es reenviarà a dos nodes si ens podem connectar amb aquella xarxa i a un únic node en cas contrari.

[28] En el nom del fitxer no s'inclou la seva extensió, que serà ".h" (per la definició de la classe i la seva estructura de dades) o ".cpp" (per la implementació).

D'altra banda, *CService* treballa sobre *CNetAddr* per tal d'emmagatzemar el port TCP (en altres paraules, la classe representa una "adreça TCP"). Al igual que la funció pare, també proporciona funcions de codificació.

Pel que respecta a la classe *CAddress*, hereta el descrit anteriorment ampliant-ne les funcionalitats amb dos característiques prou rellevants: permet conèixer els serveis oferts pel node associat a l'adreça i emmagatzema una marca de temps per conèixer quan s'ha vist per últim cop l'adreça (ja sigui de forma directa o rebuda per part d'un tercer).

Per últim, *CAddrInfo*, que hereta tot el que hem vist fins ara, emmagatzema informació per gestionar l'adreça com per exemple: l'origen (qui ens ha facilitat l'adreça), el número d'intents de connexió sense èxit i quan es va realitzar l'últim intent de connexió al node.

Amb aquestes quatre estructures bàsiques podem procedir a analitzar el següent component necessari per entendre el funcionament de la xarxa: el gestor d'adreces.

5.5 AddrMan: gestió, emmagatzemament i selecció d'adreces

Per tal de gestionar les adreces de la xarxa que acabem de veure, Bitcoin Core implementa una agenda on guarda les direccions dels nodes que va descobrint. Aquest sistema no es limita a l'emmagatzemament de les adreces, sinó que també les classifica i en selecciona les més apropiades per obrir una nova connexió.

En els seus inicis el programari implementava la seva agenda en una base de dades convencional, el que comportava una sèrie de inconvenients[29]. No obstant, a partir del 22 de març de 2012, en el commit 5fee401 [17], es va introduir una agenda d'adreces estocàstica, que serà la que analitzarem al llarg d'aquest apartat.

Un sistema estocàstic o no determinista és un mecanisme que funciona amb un comportament totalment o parcialment incert. En aquest sentit l'agenda implementa una certa aleatorietat en el moment de prendre decisions clau, com ara la selecció d'adreces, la seva classificació o l'eliminació d'adreces antigues.

El mecanisme implementat per l'agenda és complex i, conseqüentment, el descriurem per seccions: primer de tot exposarem les estructures de dades que empra, tot seguit continuarem descrivint el mecanisme d'incorporació d'adreces, en tercer lloc parlarem breument sobre el seu manteniment i, per últim, analitzarem la selecció d'adreces.

5.5.1 Addrman: estructura de dades

El gestor d'adreces es troba implementat per la classe *CAddrMan*, el seu codi resideix en *addrman.h* i *addrman.cpp* i, tal com s'indica en el fitxer capçalera, té les següents estructures de dades:

Nom	Tipus	Descripció o funcionalitat
nKey	256 bits BLOB	Clau secreta emprada per l'aleatorietat en la selecció d'adreces.
nIdCount	enter	Comptador dels identificadors de les adreces de l'agenda.
mapInfo	Mapa <enter, CAddrInfo>	Mapa que relaciona els identificadors de cada adreça amb la pròpia adreça.
mapAddr	Mapa <CNetAddr, enter>	Mapa que relaciona una adreça amb el seu identificador.

[29] Com per exemple l'elevat número d'escriptures al disc i la lentitud en la gestió d'adreces [26].

Nom	Tipus	Descripció o funcionalitat
vRandom	Vector<enter>	Vector ordenat aleatòriament amb tots els identificadors.
vvTried	Matriu<enter> <enter>	Matriu de les adreces amb les que s'ha establert una connexió,
nTried	enter	Número d'adreces en la taula anterior.
vvNew	Matriu<enter> <enter>	Matriu de les adreces noves, amb les que no s'ha establert una connexió (o bé,
nNew	Enter	Número d'adreces en la taula anterior.

Taula 16: estructura de dades de l'agenda d'adreces (*AddrMan*)

De la taula a dalt exposada emfatitzarem les següents variables:

vvTried i *vvNew* emmagatzemen, respectivament, les adreces amb les que s'ha establert connexió ("adreces provades") i amb les que no s'ha establert una connexió[30] ("adreces noves"). Aquestes dues matrius representen un conjunt de *sacs* (o *buckets* en anglès) on s'emmagatzemen desordenadament les adreces. El seu número està establert en 1024 sacs d'adreces noves i 256 sacs de provades, on cada un d'ells pot arribar a contenir 64 adreces.

nKey és una variable ideada per proporcionar entropia en els mètodes anteriors de selecció dels sacs. En particular la variable empra la funció *RAND_bytes*[31] per iniciar-se aleatòriament en la construcció de l'objecte *CAddrMan*. Dita variable s'empra en la selecció d'un sac i d'una posició en un sac (funcions *GetTriedBucket*, *GetNetBucket* i *GetBucketPosition*) tal com es mostra a continuació:

```
int CAddrInfo::GetTriedBucket(const uint256& nKey) const
{
    uint64_t hash1 = (CHashWriter(SER_GETHASH, 0) << nKey <<
                                                    GetKey()).GetHash().GetCheapHash();
    uint64_t hash2 = (CHashWriter(SER_GETHASH, 0) << nKey << GetGroup() <<
            (hash1 % ADDRMAN_TRIED_BUCKETS_PER_GROUP)).GetHash().GetCheapHash();
    return hash2 % ADDRMAN_TRIED_BUCKET_COUNT;
}
```

Codi 4: exemple d'ús d'*nKey* en Bitcoin Core

mapInfo i *mapAddr* són dues matrius que relacionen, respectivament, els identificadors d'una adreça amb la pròpia adreça (representada per l'objecte *CAddrInfo*) i una adreça (representada aquest cop per *CNetAddr*) amb el seu identificador.

Per últim, *vRandom* és una taula que s'empra per emmagatzemar aleatòriament els identificadors de les adreces. En aquest sentit, cada cop que es crea una adreça en l'agenda (funció *CAddrMan.Create*) s'introdueix el nou identificador al final d'aquesta variable i, quan és necessari obtenir diverses adreces de l'agenda (mètode *CAddrMan.GetAddr_*), es desordena parcialment (funció *CAddrMan.Swap*) i s'utilitza per seleccionar una direcció aleatòria.

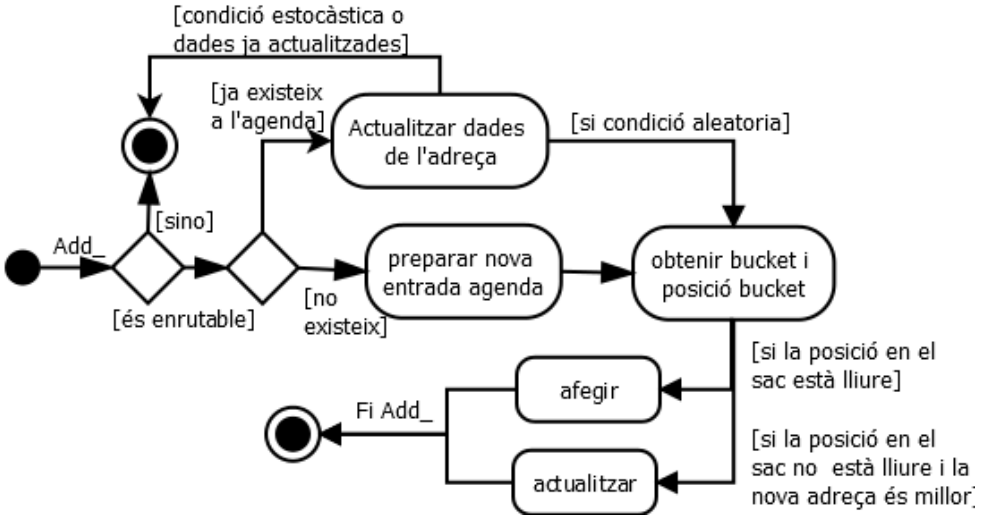
Són aquestes estructures on s'afegiran les adreces, tal com exposem en el següent apartat, es gestionaran (apartat 5.5.3), i es seleccionaran (apartat 5.5.4).

[30] Tal com veurem en 5.5.3 també emmagatzema adreces que ja s'han provat.
[31] Funció d'OpenSSL https://www.openssl.org/docs/manmaster/crypto/RAND_bytes.html [data de consulta 2015-10-26].

5.5.2 Addrman: mecanisme d'incorporació d'adreces

Tal com acabem de veure, Bitcoin Core, organitza les adreces en dos grans blocs: les "adreces provades" i les "adreces noves". Les primeres són aquelles amb les que s'ha pogut establir una connexió (i per tan són vàlides) mentre que les segones són aquelles que no s'han provat però es coneixen perquè un igual de la xarxa ens les ha facilitat.

En un principi, quan el programari rep una nova adreça (independentment de com l'hagi rebut) invoca a la funció *Add* de la classe *CAddrMan* per tal d'incorporar-la a l'agenda o actualitzar-ne les dades si ja existia. Dita funció realitza les següents accions[32] (totes es realitzen sobre el sac "d'adreces noves"):



Imatge 13: diagrama d'activitats per afegir una adreça de xarxa a Bitcoin Core.

Com podem veure en el diagrama anterior el comportament de la funció *Add_* és lleugerament diferent segons si l'adreça existeix a l'agenda. En aquest sentit si es tracta d'una nova direcció, es crea una nova entrada i es selecciona aleatòriament un sac i una posició dintre seu. Si la posició està buida l'adreça s'afegeix, en canvi si la posició ja està ocupada es comprova quina de les dues adreces és millor i es conserva aquesta.

D'altra banda, si es tracta d'una adreça que ja existia el procediment és lleugerament més complex:

A. Es recupera la informació corresponent a la citada direcció (funció *CAddrMan*.*Find*).
B. S'actualitza la informació de l'adreça: l'últim cop que s'ha "vist" (si la data és més nova de la que figurava) i els serveis que ofereix el node associat a la direcció.
C. Es realitza una comprovació estocàstica: la probabilitat d'afegir una adreça per primer cop és u (P = 1) i, però per cada cop que s'intenta afegir de nou, la probabilitat disminueix en 0.6.

En aquest últim cas ens podem trobar que una mateixa adreça aparegui referenciada diversos cops en diferents posicions d'un o diversos sacs amb el màxim definit en el programa: 8 còpies de l'adreça.

[32] La funció *Add* pot tractar diverses adreces alhora i, per cada una d'elles, invoca a la funció *Add_*

En la selecció de les adreces també participa un mecanisme d'agrupació, que classifica les adreces d'acord al subrang /16 de l'adreça i, a partir d'aquest rang, s'obtenen aleatòriament 64 cubells.

5.5.3 Addrman: manteniment de les adreces

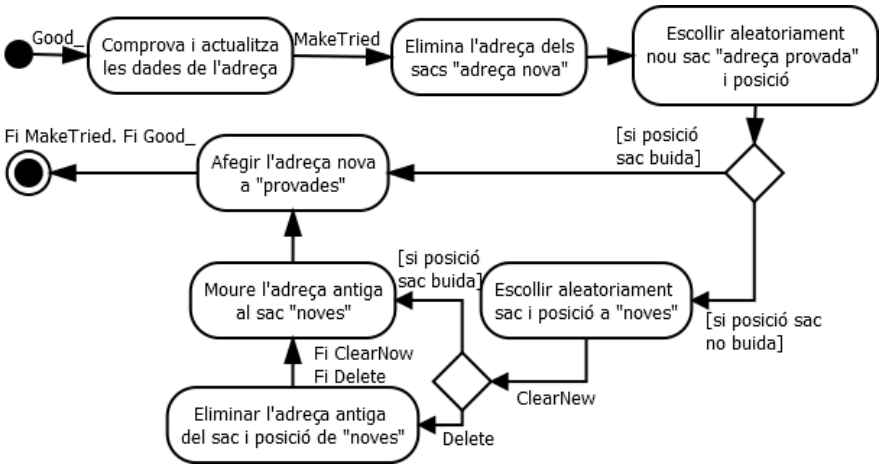
El següent punt a tractar és el manteniment de les direccions de l'agenda de Bitcoin Core. Amb manteniment ens referim a les tasques d'organització de les adreces (la classificació entre les "adreces provades" i les "noves" i com passen d'un estat a l'altre), eliminació de les adreces antigues o dolentes i la seva conservació en base de dades.

Pel que respecta al primer punt, l'**organització** de les adreces, ens podem trobar en dues situacions: que una "adreça nova" passi a ser una "adreça provada", que succeirà quan el client estableixi una connexió; i el cas invers, que una "adreça provada" passi a ser una "adreça nova", quan la primera sigui substituïda per una adreça millor (tal com hem vist en el diagrama anterior, la Imatge 13).

Les dues situacions es troben implementades en el programari mitjançant la funció *Good_* de la classe C*AddrMan*. La funció comprova si l'adreça a moure realment és una "adreça nova" i, si és així, invoca a la funció *MakeTried* de la mateixa classe per tal de marcar la direcció "nova" com a "provada".

Aquest procés (tal com veurem en el proper diagrama) implicarà els següents passos:

A. Es comprovarà si l'adreça es pot moure als sacs de "provades" i, addicionalment, se n'actualitzarà les dades.
B. Esborra l'adreça de totes les posicions de tots els sacs de direccions "noves". Selecciona una posició i en els sacs de "provades".
    1. Si la posició està buida l'adreça l'ocuparà.
    2. Si la posició no està buida es l'adreça l'ocuparà igualment i la direcció que hi estava abans es mourà al sac de "noves". En el aquest procés de moure la direcció al sac de les "noves" succeirà el mateix: s'obtindrà una posició i, si aquesta posició està buida, l'adreça si mourà; però si la posició no està buida, es compararan les dues adreces "candidates" i si quedarà la millor.



Imatge 14: diagrama d'activitats per marcar una adreça de xarxa com a provada.

Pel que fa el procés d'**esborrat**, indicat en el punt C.2, cal tenir present que una adreça pot estar en més d'una posició en més d'un sac i, . per tant, "esborrar" una adreça no

implica eliminar-la per complet de l'agenda, sinó que es limita a eliminar-la de la posició concreta que ocupa en el sac.

Per últim, el manteniment també inclou marcar una adreça com "**intentada**". Quan el node intenta connectar-se a una adreça i no ho aconsegueix, augmenta en un comptador dels intents de connexió sense èxit a una direcció en concret. Aquest valor té rellevància més endavant, en la selecció de les adreces.

5.5.4 Addrman: selecció d'adreces

Un altre aspecte rellevant per comprendre el funcionament de la xarxa Bitcoin és analitzar com selecciona l'agenda d'adreces del programari una direcció quan se li sol·licita. Depenent de l'adreça que proporcioni l'agenda, el node "veurà" una part o altra de la xarxa, pel que el seu comportament en el descobriment de veïns se'n veurà afectat.

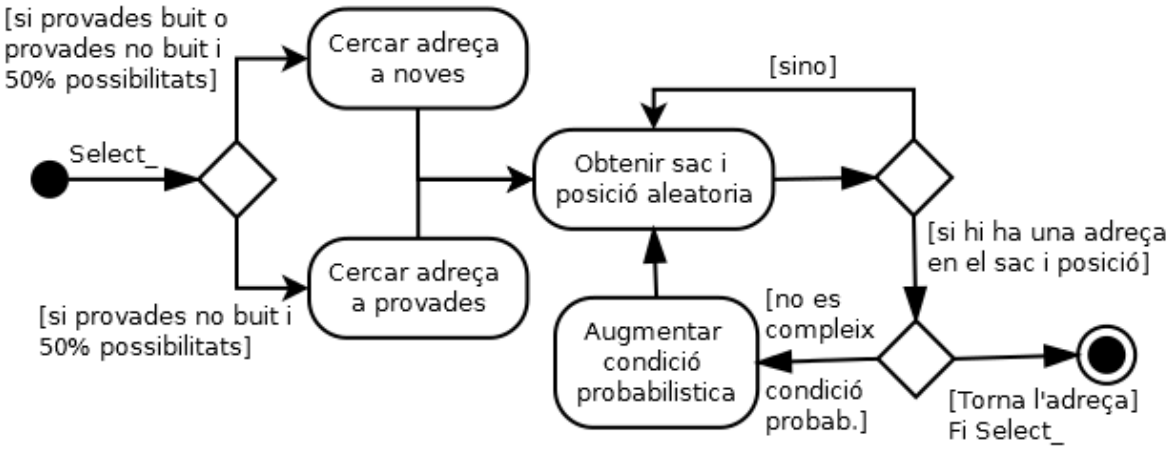
En aquest sentit cal considerar que el programari Bitcoin Core implementa dues funcions per a la selecció d'adreces: escollir una única direcció (utilitzada principalment en la selecció d'adreces per obrir una connexió de sortida) o escollir alhora diverses adreces (emprada en la resposta d'un missatge *addr*).

Com que ambdues funcionalitats tenen un comportament lleugerament diferent les exposarem per separat.

5.5.4.1 Selecció d'una única adreça

Tal com hem vist anteriorment, l'agenda de Bitcoin Core s'implementa sobre un sistema estocàstic. A l'escollir una adreça amb aquest sistema existeix una certa aleatorietat en la selecció, tot i que també té pes en la decisió la "reputació" de l'adreça (l'últim moment en que "s'ha vist" o el número d'intents de connexió sense èxit).

En aquest sentit el codi que determina les funcionalitats de selecció d'una direcció es troba implementat en la classe *CAddrMan* i, en particular, en la funció *Select_* escrita en els fitxers *src/addrman.h* i *src/addrman.cpp*. El funcionament és el descrit en el següent diagrama



Imatge 15: diagrama d'activitats per seleccionar una adreça de xarxa en Bitcoin Core.

Primer de tot, com podem veure, es pren una decisió estocàstica per determinar en quina de les dues matrius es va a cercar la direcció: en la matriu amb els sacs corresponents a les adreces “provades” o a la matriu amb les adreces “noves”. Aquesta decisió es pren depenent de si alguna matriu està buida o, si no ho estan, amb P = 0.5. Tot seguit, en la matriu seleccionada, la funció escull aleatòriament un sac i una posició aleatòria. Si en aquell sac i posició concretes hi ha una adreça llavors es procedeix a una segona comprovació aleatòria:

```
if (GetRandInt(1 << 30) < fChanceFactor * info.GetChance() * (1 << 30))
```

Codi 5: condició probabilística de selecció d’una adreça de xarxa en Bitcoin Core.

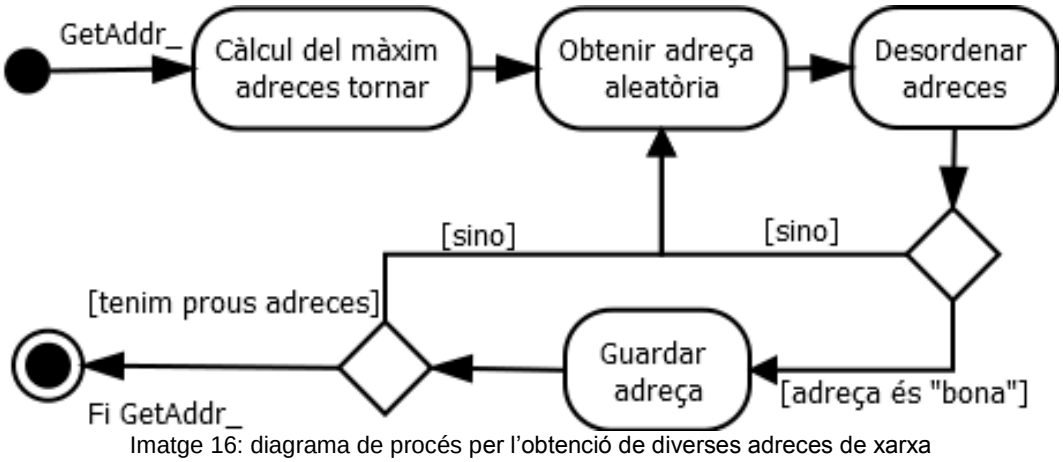
En el bocí de codi anterior, que implementa la condició estocàstica, trobem els següents elements:

- La variable *fChanceFactor* inicialment es troba inicialitzada en 1,0 i augmenta el seu valor 1,2 cops cada cop que la condició aleatòria descarta una adreça.
- La funció *info.GetChance()* torna un valor d’acord a la reputació de l’adreça que s’avalua en la iteració. La reputació rep una petita penalització si fa “temps” que no es tenen notícies del node, i una gran penalització per cada cop que s’hagi realitzat una connexió sense èxit.

En conclusió, podem veure com la selecció de les adreces intenta equilibrar l’oportunitat de connectar-se entre les adreces “noves” i “provades”, a la vegada que intenta maximitzar l’aleatorietat en l’elecció d’una direcció amb una petita tendència cap a les adreces amb millor reputació.

5.5.4.2 Selecció de múltiples adreces.

Al igual que en la selecció d’adreces simple, en l’obtenció de múltiples adreces també hi intervé una certa aleatorietat, tot i que en menor mesura –el que suposem que es deu per millorar el rendiment del programa ja que hem d’escollir diverses adreces. En particular la funcionalitat es troba implementada en la classe *CAddrMan*, mencionada anteriorment, i en la funció *GetAddr_*:



Imatge 16: diagrama de procés per l’obtenció de diverses adreces de xarxa

Com podem observar en la imatge anterior, el programa primer de tot determina el número màxim de direccions que ha de tornar la funció d’acord a un percentatge màxim (el 23 per cent) o bé un màxim de 2.500 adreces. Tot seguit obté una adreça aleatòria, sense diferenciar entre les direccions “provades” i “noves”.

En tercer lloc –suposem que per millorar l'aleatorietat– canvia la posició entre la adreça que hem seleccionat i una posició aleatòria (mitjançant el vector *vRandom*, comentat en l'apartat 5.5.1). A continuació, la funció avalua si l'adreça "és terrible" (una adreça terrible és aquella impossible, que fa molt de temps de la que no es sap res o hi ha hagut masses intents de connexió sense èxit) per descartar-la.

Per últim, si l'adreça és "bona" l'emmagatzema i, quan tenim totes les necessàries les retorna.

5.6 Gestió de les connexions

En aquest apartat exposarem el funcionament bàsic del programari Bitcoin Core pel que respecta a la seva interacció amb la xarxa de parells. Per aconseguir-ho, descriurem com el programa inicia la xarxa des de que s'arranca, el que ens permetrà tenir una visió global de tots els mecanismes que hi participen.

No obstant, abans de començar, hem de detallar les estructures més importants pel que fa les connexions: *CNode* i *vNodes*. El primer emmagatzema les dades necessàries per mantenir les connexions (a continuació en mostrem els atributs principals) i el segon és una llista dels nodes (és a dir objectes *CNode*) amb els que s'ha establert una connexió.

En relació a aquest punt, cal recordar que quan s'estableix una nova connexió, l'adreça del node amb el que ens connectem passarà, en l'agenda de direccions, al sac de les "adreces provades" tal com hem comentat anteriorment en l'apartat 5.5.3).

Nom	Tipus	Descripció
nServices	uint64_t	Serveis oferts per el node.
hSocket	Socket	
vSendMsg	BLOB	Cua de missatges per enviar al node[33].
vRecvMsg	BLOB	Cua de missatges rebuts del node.
nLastSend	int64_t	Últim cop que s'ha enviat un missatge.
nLastRecv	int64_t	Últim cop que s'ha rebut un missatge.
nTimeConnected	int64_t	Temps de connexió (connexió inicial).
nTimeOffset	Int64_t	Diferència de temps.
addr	CAddr	Adreça del node.
addrLocal	CService	Adreça local del node.
nVersion	Enter	Versió inicial.
nRefCount	Enter	El número de connexions establertes amb aquest node.
Id	Enter	Identificador intern del node.

Taula 17: principals atributs d'un igual en Bitcoin Core

Abans de continuar també haurem d'analitzar la gestió de les connexions. Bitcoin Core, al igual que qualsevol client d'una xarxa d'iguals, podrà realitzar **connexions "de sortida"**

[33] Ampliem la informació sobre les cues de missatges i els marcadors (*flags*) en el tercer annex.

(*outbound*) i rebrà **connexions "d'entrada"** (*inbound*). En aquest sentit el número de connexions màximes del programari està limitat per tres condicions:

- DEFAULT_MAX_PEER_CONNECTIONS. La variable, que es troba definida en el fitxer *net.h*, limita el número de connexions màximes (tant d'entrada com de sortida) que pot realitzar el programa. El límit actual està establert en 125.
- MAX_OUTBOUND_CONNECTIONS. Constant, definida en el fitxer *net.cpp*, i que limita el número de connexions de sortida màximes. El límit actual s'estableix en 8.
- A més a més, existeix una tercera variable per controlar el número connexions entrants. Dita variable és la diferència entre les connexions màximes i les connexions de sortida (pel que el programari permet un màxim de 117 connexions d'entrada).

Quan el programari vol realitzar una connexió, ja sigui d'entrada o sortida, compararà el número de connexions respectives amb el seu màxim. Si s'ha assolit el límit, no es permetrà la connexió (es refusaran les connexions d'entrada o no es realitzarà una de sortida), en cas contrari es realitzarà la connexió.

Tècnicament, Bitcoin Core controla el número de connexions d'entrada comparant les connexions actives amb el seu límit (funció *AcceptConnection* que s'invoca des del mètode *ThreadServiceSocket* que veurem a continuació); pel que fa les connexions de sortida es controlen amb un semàfor: s'inicialitza el semàfor amb el número màxim de connexions de sortida i no es permet continuar el procés de connexió si s'ha invocat el semàfor més cops dels autoritzats.

```
CSemaphoreGrant grant(*semOutbound);
boost::this_thread::interruption_point();
```

Codi 6: semàfor que limita les connexions de sortida en Bitcoin Core.

Amb tota aquesta informació, ja podem afrontar la inicialització de serveis del programari: Bitcoin Core inicialitza els serveis de xarxa des del fitxer *scr/init.ccp* mitjançant una crida a la funció *StartNode*, que realitza els següents passos:

1) Invoca a l'agenda d'adreces per llegir-les de la base de dades (si n'hi ha).
2) Llegeix la llista de nodes prohibits de la base de dades (*banned*).
3) Determina el número de connexions de sortida màximes.
4) Obté la IP local (de rellevància si el node disposa d'una adreça IP privada).
5) Es creen cinc fils d'execució periòdica per donar suport a les funcionalitats de xarxa.
6) Inicia una tasca periòdica per anar emmagatzemant les adreces conegudes pel node en base de dades.

Dels punts anteriors únicament en centrarem en el cinquè, ja que és el més rellevant per la gestió de les connexions. En aquest sentit els cinc fils que es creen i s'executen periòdicament són els següents:

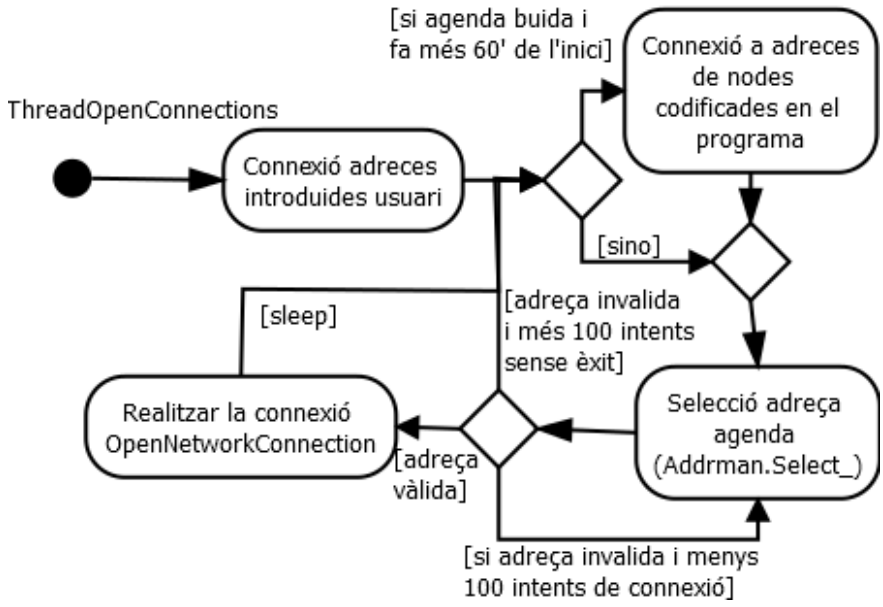
- *ThreadDNSAddress*: el fil es connectarà a diversos serveis DNS per obtenir iguals si es compleixen una de les següents condicions: l'usuari ha indicat l'obligatorietat d'obtenir iguals mitjançant la consulta de DNS, no hi ha connexions obertes o l'agenda està buida.
- *ThreadSocketHandler*: realitza el manteniment dels sockets. En concret gestiona les desconnexions dels nodes que no responen o amb els que no s'està treballant,

accepta noves connexions d'entrada (si no es supera el límit) i, per últim, rep i envia informació pels sockets.

- *ThreadOpenAddedConnections*: si l'usuari ha introduït manualment una llista de nodes als que connectar-se aquest fil intenta la connexió.
- *ThreadMessageHandler*: per cada node amb el que s'ha establert una connexió, el fil processa els missatges obtinguts per *ThreadSocketHandler* i prepara els missatges per tal que *ThreadSocketHandler* els enviï.
- *ThreadOpenConnections*: aquest fil s'encarrega de realitzar les connexions de sortida del node (sempre que no es superi el límit). Com que el seu funcionament és rellevant per la fi del projecte n'exposarem el seu funcionament en els propers paràgrafs.

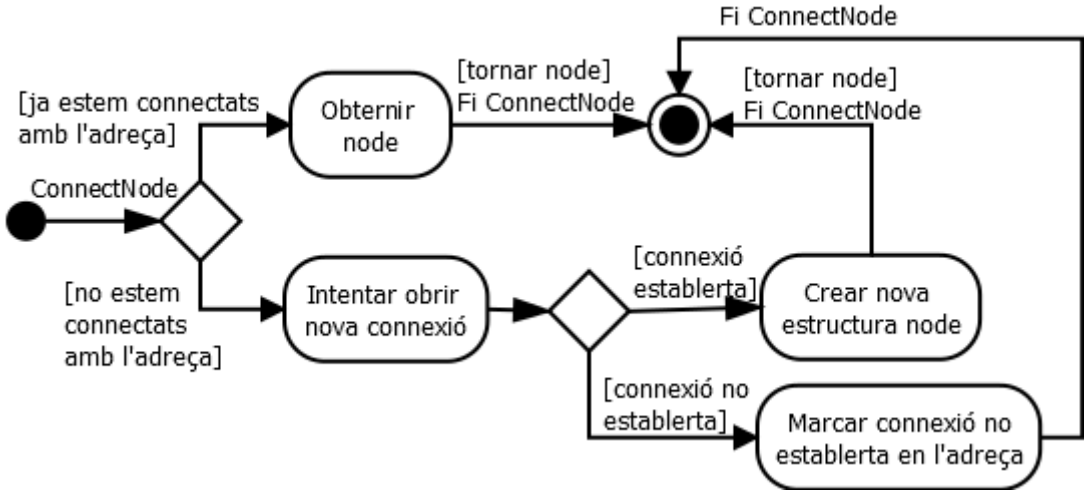
Tal com comentàvem, el fil *ThreadOpenConnections*, s'encarrega de les connexions de sortida del node. Per realitzar aquesta tasca ha d'obtenir una adreça on connectar-se i establir la connexió en si. L'obtenció de l'adreça es realitza en la funció que implementa el fil, mentre que la connexió s'implementa en la funció *OpenNetworkConnection*.

Per obtenir adreces *ThreadOpenConnections* en primer lloc, intenta connectar-se als nodes introduïts per l'usuari. Tot seguit comprova si hi ha adreces a l'agenda, si no n'hi ha intenta connectar-se als iguals codificats en el mateix programa. Sigui com sigui, a continuació el fil entra en un bucle on intentarà connectar-se a adreces de l'agenda. El seu funcionament s'exposa en el següent diagrama:



Imatge 17: diagrama de procés del fil que obre connexions en Bitcoin Core

OpenNetworkConnection és una segona funció que, conjuntament amb *ConnectNode* i tal com el seu nom indica, prepararà una connexió de sortida amb un igual. Per establir la connexió la funció *ConnectNode* realitza les següents tasques:



Imatge 18: diagrama de procés per la connexió amb un node

En cas que no es pugui establir una connexió es marcarà l'adreça indicant-ho. Així, tal com hem vist abans en el apartat 5.5.4.1, aquest intent sense èxit es tindrà en consideració en la reputació de les adreces, que entrarà en joc quan l'agenda seleccioni un altre cop la direcció.

5.7 Conclusions

Tal com hem pogut veure al llarg d'aquest capítol, el procés de descobriment dels veïns del client Bitcoin Core compren diversos mecanismes i tecnologies. En aquest sentit, per tal d'assegurar que el simulador BTC-NS reprodueixi de forma fidel les funcionalitats del programari, caldrà que implementi les característiques enumerades en la següent taula.

Id.	Categoria	Subcategoria	Secció document	Aplicabilitat al simulador
BC-01	Agenda	Addició	5.5.1	Haurà d'implementar un mecanisme que permeti dividir l'emmagatzematge d'adreces entre "provades" i "noves".
BC-02	Agenda	Addició	5.5.1	Haurà d'implementar un mecanisme de sacs que permeti aleatoritzar el procés.
BC-03	Agenda	Manteniment	5.5.3	Permetre marcar una adreça de l'agenda com a "provada" i a l'inrevés.
BC-04	Agenda	Selecció	5.5.4.1	Implementar un mecanisme per seleccionar una adreça de l'agenda, equilibrant la selecció de direccions "noves" i "provades" considerant la reputació de la mateixa.
BC-05	Agenda	Selecció	5.5.4.2	Implementar un mecanisme per seleccionar diverses direccions de l'agenda amb un procediment aleatori
BC-06	Connexió	-	5.3.1	Els nodes hauran d'implementar un mecanisme per obtenir adreces d'un servidor DNS

Id.	Categoria	Subcategoria	Secció document	Aplicabilitat al simulador
BC-07	Connexió	-	5.3.1	Els nodes hauran de disposar adreces per defecte codificades en el propi programa.
BC-08	Connexió	-	5.5.4.1	Els nodes hauran de consultar a l'agenda per demanar-li una adreça on connectar-se
BC-09	Missatges	Adreces	5.3.2	Els nodes hauran de poder enviar missatges *getaddr* i respondre amb missatges *addr*, obtenint les adreces de l'agenda mitjançant una selecció múltiple.
BC-10	Descob. veïns	Bootstrap	5.3.1	Ens nodes han d'implementar mecanismes d'inici i connexió a la xarxa.
BC-11	Descob. Veïns	-	5.3.2	Els nodes s'han d'anunciar periòdicament als nodes amb els que han establert una connexió.
BC-12	Descob. Veïns	-	5.3.2	En l'inici de la connexió de dos nodes, el node que inicia la connexió s'anunciarà.
BC-13	Descob. Veïns	-	5.3.2	Els nodes hauran de reenviar les adreces que reben a uns nodes determinats.
BC-14	Connexió	-	5.1	Els nodes han d'implementar el mecanisme de connexió en tres missatges.

Taula 18: funcionalitats de Bitcoin Core pel descobriment de veïns

A partir d'aquesta taula, en el capítol 7 podrem analitzar i implementar les millores necessàries al simulador. Per realitzar-ho, prèviament caldrà determinar el funcionament del simulador BTC-NS que analitzarem en el següent capítol.

6 L'eina de simulació: el BTC-NS

Per assolir l'objectiu del projecte cal que l'eina sobre la que es realitzin les simulacions de la xarxa d'iguals Bitcoin, el BTC-NS, tingui un comportament *fidedigne* respecte a la xarxa real pel que respecta al descobriment de veïns. Per aconseguir-ho caldrà comparar el funcionament actual del simulador (que desenvoluparem al llarg d'aquest capítol) amb el funcionament del món real (mitjançant l'estudi de l'eina Bitcoin Core que hem vist en el capítol anterior), identificant i implementant en el marc del projecte les millores necessàries.

Per procedir a l'estudi del BTC-NS, l'eina de simulació, procedirem a analitzar els següents aspectes:

A. Les opcions de simulació permeses en el programari, en el benentès que caldrà determinar posteriorment quina és la millor opció per dur a terme l'estudi de l'eficàcia de l'aranya.
B. Els tipus de nodes implementats. En aquest sentit, per l'estudi hi intervindran un mínim de 3 nodes diferents: els nodes "convencionals", els nodes "DNS" (que proporcionaran serveis d'inici de la xarxa) i les aranyes (per descobrir la topologia de la xarxa).
C. L'arquitectura dels nodes, o en altres paraules: com estan organitzades les diferents classes i mètodes implicats en el funcionament d'un node en el BTC-NS (limitant-nos a la seva invocació sense analitzar-ne el funcionament).
D. El mecanisme de comunicació en el simulador. Cal conèixer com els nodes estableixen les connexions, gestionen la seva desconnexió després d'un temps determinat, seleccionen i gestionen les adreces d'altres iguals en la xarxa i es comuniquen entre si per missatgeria.
E. La inicialització de la xarxa en els nodes. Aquest apartat inclouria tant l'estudi de com els nodes inicialitzen la xarxa d'iguals com les funcionalitats proporcionades pels nodes DNS.
F. El funcionament de l'aranya.
G. Les dades que emmagatzema el simulador (limitant-nos a les necessàries per l'estudi de l'eficàcia de l'aranya).
H. Per últim exposarem breument SimPy, la biblioteca que permet executar les simulacions.

Pel que fa l'abast d'aquest anàlisi, no inclourà l'estudi dels següents components:

A. La tecnologia emprada en la programació del simulador.
B. L'estat d'aquelles funcionalitats que divergeixin de l'àmbit del descobriments de veïns i de la xarxa d'iguals com, per exemple, la cadena de blocs.
C. L'emmagatzemament en base de dades dels components que no formin part de la xarxa.
D. L'anàlisi es du a terme a partir de la versió 229 del codi emmagatzemant en el repositori Subversion del simulador (obtingut a mitjans d'octubre de 2015).
E. No es consideraran els nodes amb adreces privades i accedint a la xarxa mitjançant un sistema NAT.

6.1 Funcionalitats del simulador

Tal com veurem a continuació, el BTC-NS no proporciona un únic mecanisme per realitzar les simulacions, sinó que proporciona diverses opcions per tal de dur-les a terme. En concret disposem de quinze tipus de nodes, diversos mètodes per gestionar l'evolució de la xarxa i mecanismes per simular components de Bitcoin [18].

Pel que fa als tipus de nodes, de tots els existents únicament en tractarem tres: *crawler*, que serà el node encarregat d'actuar com una aranya; *DNS seed*, que s'encarregarà de proporcionar un servei d'inicialització a la xarxa dels altres nodes; i, per últim, nodes *convencionals* que participaran com a "clients de la xarxa"[34].

En relació a l'evolució de la xarxa d'iguals, el simulador proporciona tres possibilitats: *xarxa estàtica*, on totes les connexions es defineixen des d'un inici i no canvien al llarg de la simulació; *xarxa dinàmica definida*, els nodes i les seves connexions canvien d'acord a una programació específica; i el descobriment de veïns, on la xarxa és dinàmica[35].

Respecte a la creació dels propis nodes, en la xarxa estàtica es creen des d'un inici; en la xarxa dinàmica definida, els nodes es creen d'acord al que s'hagi definit en la simulació i, per últim, en la xarxa dinàmica no s'ha definit quan i com es creen els nodes, pel que serà una de les tasques a realitzar.

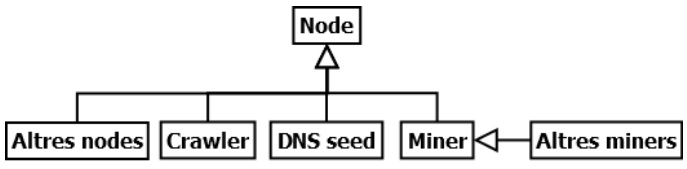
A banda de l'anterior, també es poden establir els models de latència i ample de banda de les connexions entre nodes en el simulador. En concret es contemplen tres models per la latència i l'ample de banda: constant, que no varia; uniforme, valors aleatoris d'acord a una distribució uniforme; i *normal*, valors aleatoris d'acord a una distribució normal.

També cal considerar com inicialitza el programa els valors dels diferents nodes que participaran en la simulació. Per obtenir aquesta informació, el simulador implementa dos sistemes: el normal, que permet a l'usuari definir una entorn mitjançant una interfície web; i l'avançat, que permet definir un escenari concret (és a dir, cada un dels nodes) mitjançant un fitxer GML[36].

Per últim, el simulador implementa diverses opcions per ignorar determinats mecanismes de la xarxa Bitcoin i determinar com és comporten. Per exemple, es podria ignorar la validació dels blocs i que, davant d'un bloc els nodes sempre els consideressin vàlid (o invàlid). Pel que respecta al projecte podem separar els mecanismes que tenen relació amb la xarxa d'iguals (que afectaran al projecte) i la resta.

6.2 Els nodes en el simulador

Tal com indicàvem, com a mínim el simulador haurà de disposar de tres tipus de nodes: *l'aranya*, els *DNS seed* i els *nodes convencionals*. Tots els nodes es troben implementats en el fitxer *node.py* basant-se en un mecanisme d'herència: la classe *Node* implementa el comportament bàsic que es personalitza en els nodes que es deriven tal com es veu a continuació:



Imatge 19: estructura d'herència en el simulador BTC-NS.

Per tant, aquesta classe implementarà bona part de les funcionalitats de xarxa que la resta de nodes compartiran. En el següent apartat ens dedicarem a analitzar, en primer lloc, la seva l'arquitectura (emmarcant-nos en les funcionalitats de xarxa), posteriorment

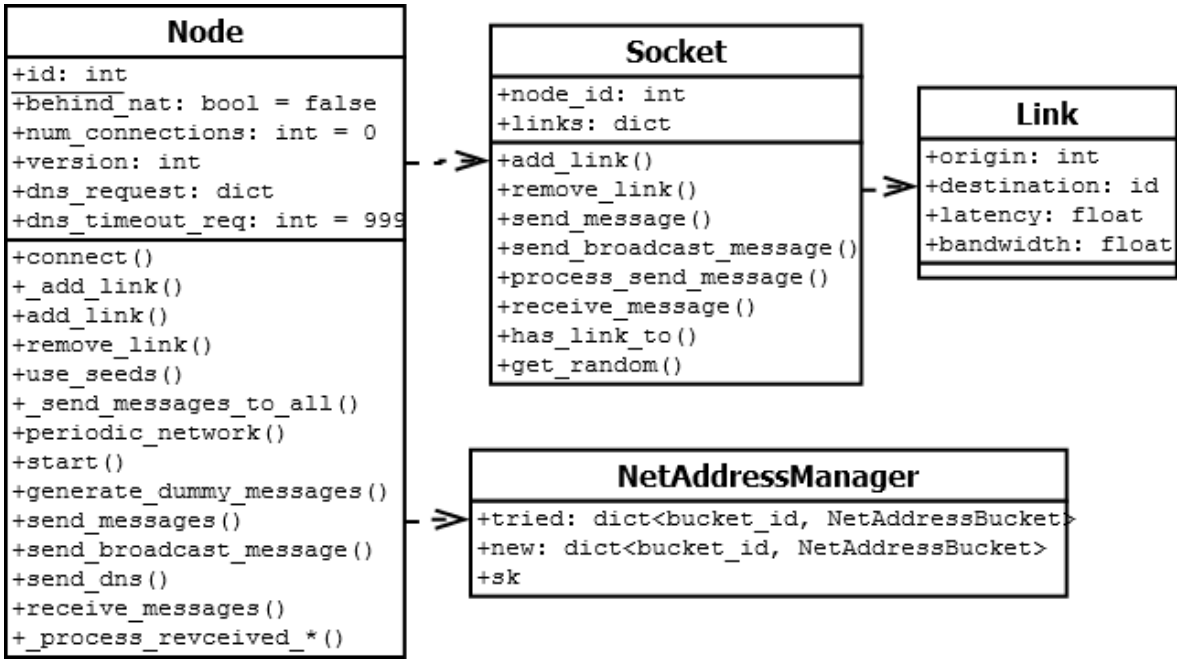
[34] Al interessar-nos únicament les funcions de xarxa podem prescindir dels altres components.
[35] En el benentès que, a diferència de les altres opcions, no es coneix quina serà l'evolució de la xarxa, ja que dependrà del descobriment de veïns dels nodes.
[36] GML és un acrònim de *Graphic Modelling Language* [1].

procedirem a analitzar les diferències dels altres dos nodes (si existeixen) i, al final del document, en el capítol 9.1, exposarem possibles millores en el seu disseny.

6.2.1 Arquitectura dels nodes

Entenem amb arquitectura el disseny i estructuració del codi, ja sigui en variables, mètodes, tipus de programació, classes, patrons o APIs. Per tant, en aquest apartat procedirem a analitzar com estan estructurades les funcionalitats en els nodes del simulador i com implementen i estructuren la xarxa.

Per aconseguir-ho, començarem analitzant la classe *Node*. Com podem veure en el següent diagrama, usa tres classes addicionals: *Socket*, encarregada de la gestió de les connexions del node*; Link*, que representa una connexió; i *NetAddressManager*, que implementa la gestió de les adreces conegudes pel node.



Imatge 20: arquitectura de xarxa d'un node en el BTC-NS.

En el diagrama de classes anterior també podem veure les variables i les funcions, relacionades amb la xarxa de totes les classes (excepte *NetAddressManager* que, degut a la seva complexitat, tractarem per separat més endavant). Amb aquesta informació podem donar el següent pas, i procedir a estudiar la implementació de la xarxa.

Documentarem l'arquitectura de xarxa en dues parts: d'una banda dibuixarem els processos corresponents a la xarxa, agrupant-los en connexions (establiment, manteniment, tancament i descobriment de veïns), adreces (alta i gestió) i missatges (enviament i recepció) i, d'altre banda, analitzarem els nodes *DNS* i *Crawler*.

6.2.1.1 Implementació de les connexions

Les connexions entre nodes s'implementen d'acord al model de connexió escollit en el simulador. En concret, en els models *xarxa estàtica* o *xarxa dinàmica definida*, les connexions estan gestionades pel simulador (pel que els nodes no creen, mantenen o tanquen connexions) ja que aquestes es coneixen des d'un inici.

No obstant, en el model de *descobriment de veïns* el node sí que ha de disposar d'un conjunt de funcionalitats que li permetin realitzar l'inici de la xarxa, descobrir veïns i

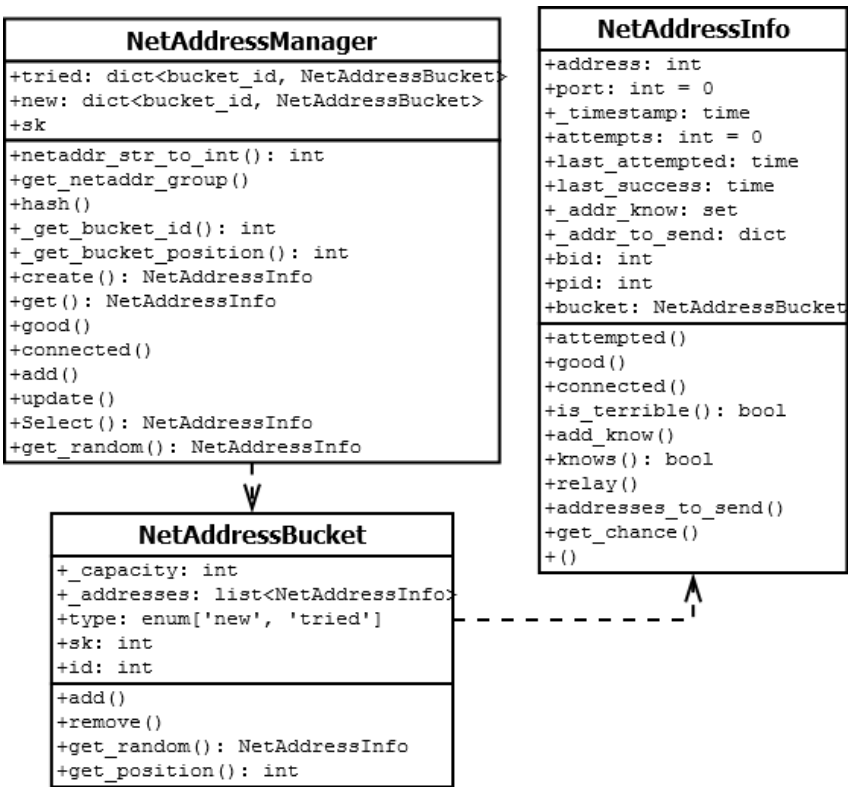
realitzar, mantenir i tancar una connexió. Actualment aquestes funcionalitats no estan implementades en la classe *Node* o en el simulador[37].

Per últim, d'acord a les especificacions del BTC-NS [18], tota connexió entre dos nodes té dues propietats: l'ample de banda i la latència, que representen la *qualitat* de la connexió entre dos iguals. Aquests atributs no són bidireccionals, és a dir, els valors de la connexió de *A* i el *B* poden ser diferents que els valors de la connexió de *B* amb *A*.

6.2.1.2 Implementació del gestor d'adreces

El gestor d'adreces està implementat en les classes *NetAddressManager*, *NetAddressBucket* i *NetAddressInfo* del fitxer *network.py*. La primera classe s'encarrega de la gestió de les adreces en diferents sacs, la segona implementa un dels sacs i la tercera descriu la informació que s'emmagatzemarà en una posició d'un sac.

La implementació del gestor d'adreces és molt similar al que empra Bitcoin Core descrita en el capítol 5.5, pel que, a partir del següent diagrama de classes, detallarem aquells aspectes que considerem que divergeixen respecte al client estàndard de Bitcoin.



Imatge 21: diagrama de classes de l'agenda d'adreces del BTC-NS

Del diagrama destacarem les següents característiques de l'agenda d'adreces de xarxa:

- Està dissenyada per treballar amb adreces IP (versió 4).
- Disposa de capacitat per emmagatzemar ports (quan en el simulador no s'utilitzen).
- Implementa dues variables *_addr_know* i *_addr_to_send* que, d'acord a la descripció del codi font serveixen, respectivament, per emmagatzemar les adreces conegudes pel node i les adreces que s'enviaran al node. Aquestes variables únicament apareixen en la funció *relay* (que no està implementada).

[37] En el mètode *Network. create_network_from_models* definit en el fitxer *network.py*.

Considerem que aquestes variables no haurien de figurar en el gestor d'adreces, ja que no estan relacionades amb l'agenda de direccions (en Bitcoin Core les dues variables apareixen implementades en la classe *CNode* en lloc d'*AddrMan*).

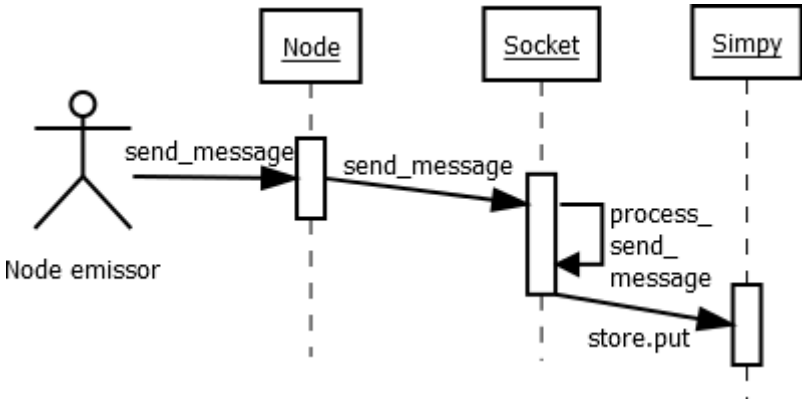
- Proporciona dues funcions per seleccionar adreces de l'agenda (*Select* i *get_random*) però no proporciona un mètode per seleccionar diverses adreces.
- Caldrà determinar si la selecció d'adreces proporciona un sistema estocàstic suficient.

6.2.1.3 Implementació del sistema de missatgeria

Una altre característica del BTC-NS, que no havíem comentat fins ara, és l'ús de la biblioteca Simpy[38]. Simpy és una eina per la simulació (que introduïm en l'apartat 6.4.1), que entre d'altres proporciona un sistema per intercanviar recursos compartits entre els elements que participen en la simulació.

En concret el simulador empra un sistema de "magatzems" [19] que permet a un component de la simulació emmagatzemar objectes per tal que un altre els recuperi. En aquest sentit és necessari un identificador que permeti discriminar a qui va dirigit l'objecte en qüestió i, a data d'avui, per intercanviar missatges s'utilitza l'identificador del node (*Node.id*).

Per enviar un missatge el simulador empra dos mecanismes: el primer és la invocació de dues funcions (*Node.send_message* o *Node.send_broadcast*) i el segon un sistema de tramesa de missatges periòdic inspirat en Bitcoin Core (el lector pot llegir l'annex III, on s'explica el seu funcionament). A continuació mostrem un diagrama de procés del primer i el codi del segon:



Imatge 22: 1r mètode de tramesa de missatges en el BTC-NS

[38] https://simpy.readthedocs.org/

```
def _send_messages_to_all(self):
    """ Send all pending messages to node (ping, getaddr, etc) """
    trickle_node = self.addrmgr.get_random()
    for node in self.addrmgr:
        if node == trickle_node:  # TODO send also if node is whitelisted
            addresses = node.addresses_to_send()
            if addresses:
                m = ds.MessageAddr(node.address, self.id, self.env.now, addresses)
                self.socket.send_message(m)

        # TODO send pings
        # TODO if initial block download rebroadcast our address
```

Codi 7: 2n mètode de tramesa de missatges en el BTC-NS

Pel que fa el segon mecanisme, sembla que aquest envia periòdicament missatges *addr* a iguals aleatoris independentment de si s'ha establert o no connexió. D'altre banda, el primer sistema, permet enviar un missatge a un node concret o fer-ne un *broadcast* i enviar un missatge a tots els nodes amb el que s'ha establert una connexió.

D'altre banda, qualsevol missatge que s'enviï per una connexió no arribarà a l'altre extrem immediatament, sinó que tardarà un temps d'acord a l'ample de banda i la latència de la connexió entre els dos nodes. No obstant, a dia d'avui, aquesta funcionalitat no està implementada.

En l'altre extrem, per rebre missatges, s'executa periòdicament la funció *Node.receive_messages* que els rep i delega el seu tractament, depenent del tipus de missatge, en un o altre mètode. Addicionalment, si algun missatge comporta resposta, es prepara i s'envia al node que l'ha sol·licitat.

En un altre ordre de coses, per comprendre el sistema de missatgeria també caldrà analitzar el funcionament i estructuració dels missatges. En concret, el simulador implementa una classe *Missatge* de la que, per herència, sorgeixen els la resta de missatges. Entre d'altres implementa els missatges *version*, *verak*, *addr*, *getaddr*, *MessageDNSQuery*, *MessageDnsResposnse* i *MessageDnsNone*.

Dels missatges anteriors, els quatre primers implementen els seus equivalents de la xarxa Bitcoin[39], mentre que els tres últims s'implementen per simular el funcionament d'un sistema DNS (pel que no són missatges propis del protocol Bitcoin). També cal destacar que implementa un tipus de missatge denominat *dummy* i que no implementa els missatges *ping* i *pong* propis de Bitcoin.

6.2.2 El sistema DNS en el simulador

Tal com hem vist en els apartats anteriors, el simulador disposa d'un tipus de node *DNS* per tal de subministrar llistats d'adreces a altres nodes. Així mateix, tal com acabem de veure, també existeixen els missatges *MessageDNSQuery*, *MessageDnsResposnse* i *MessageDnsNone* per permetre la comunicació.

El node DNS, a banda de les funcions per tractar variables, implementa un únic mètode per escoltar els missatges que rep de la xarxa i donar-los resposta. Aquest mètode únicament espera missatges *MessageDNSQuery* amb un nom de domini i, davant

[39] El lector pot trobar més informació sobre els missatges en la guia de desenvolupadors de Bitcoin [29].

d'aquest missatge, respondrà amb les adreces que coneix (*MessageDnsResposnse*) o amb un missatge d'error (*MessageDnsNone*).

```
def receive_messages(self):
  while True:
    # Wait for a network event
    if len(self.socket.links) == 0:
      return
    msg = yield self.socket.receive_messages(self.id)

    # Process the message
    if isinstance(msg, ds.MessageDnsQuery):
      if msg.domain in self.addresses:
        self.log.debug("%7.4f: Node %d will send %d addresses" %
                       (self.env.now, self.id, len(self.addresses[msg.domain])))
        answer_msg = ds.MessageDnsResponse(msg.origin, self.id, self.env.now,
                                           self.addresses[msg.domain])
      else:
        answer_msg = ds.MessageDnsNone(msg.origin, self.id, self.env.now)

    else:
      answer_msg = self._process_received_message(msg)

    if answer_msg is not None:
      self.send_message(answer_msg)
```

Codi 8: processat de missatges DNS en el simulador en el BTC-NS.

També ens hem de fixar en que es comprova si el node ha establert alguna connexió amb la xarxa Bitcoin (`len(self.socket.links) == 0`). En aquest sentit hem de tenir present que, tot i que el node forma part del simulador, no és part de la xarxa Bitcoin ni utilitza el seu protocol. En conseqüència, un node que vulgui realitzar una consulta DNS no hauria de realitzar els mateixos passos de connexió que els altres nodes.

Per tant, considerem que aquest node hauria de tenir un tractament especial en la xarxa i, conseqüentment, no hauria d'estar sotmès a les normes de comportament de la xarxa Bitcoin. En altres paraules: qualsevol node podria enviar missatges a aquest node sense la necessitat d'establir prèviament una connexió[40]. En el capítol 9.1 es proposa com a millora modificar el codi en aquest sentit.

6.2.3 L'aranya en el simulador

Tal com hem vist anteriorment, l'aranya en el simulador està implementada en la classe *BitnodesCrawler*[41] del fitxer *node.py*. El node, a banda del constructor, implementa dues funcions: *_send_discover_msg(address)*, per tal d'enviar els missatges per descobrir adreces, i *_receive_message_addr*, per processar les adreces rebudes.

[40] El funcionament proposat seria el més exacte al mon real, ja que els servidors DNS utilitzen el seu propi protocol independent de Bitcoin.
[41] Tot i que s'anomeni *Bitnodes*, no té relació amb l'aranya del món real de mateix nom.

```
def _send_discover_msg(self, address):
    """ Sends a GetAddr request to the address node.
    :param address: node address.
    :return: None
    :rtype: MessageGetAddr  # Just for  testing
    """
    self.log.debug("%7.4f: Crawler %d requests addresses to node %d" %
                   (self.env.now, self.id, address))

    msg = ds.MessageGetAddr(address, self.id, self.env.now)
    self._sent_getaddr.append(address)

    self.send_message(msg)
    return msg  # Just for Testing.
```

Codi 9: missatges pel descobriment de veïns de l'aranya en el BTC-NS

```
def _receive_message_addr(self, origin, addresses):
    """ Overwrites the Node method to process the message as a
    Crawler.

    :return: None
    :rtype: None
    """
    self.log.debug("%7.4f: Crawler %d gets %d addresses" %
                   (self.env.now, self.id, len(addresses)))

    # Process the addresses, if the address is not known
    # we ask around 'bout it.
    for k, v in addresses.iteritems():
        if k not in self.addrmgr:
            self._send_discover_msg(k)
            self._add_known_address(k, v)
```

Codi 10: processament dels missatges per part de l'aranya en el BTC-NS.

De la implementació podem observar les següents mancances:

A. La funció *_send_message* no s'executa periòdicament, pel que el node no enviarà peticions per obtenir més informació.
B. No es seleccionen a quins nodes s'enviarà el missatge.
C. Tot i que no és imprescindible es podria modificar el nom de la classe de *BitnodesCrawler* a *Crawler* per evitar confusions.
D. No s'aprecien invocacions a la base de dades, pel que es desconeix com s'emmagatzemarà la informació recollida.

6.3 Emmagatzemament de dades

Per tal de poder treure profit del BTC-NS, cal que aquest emmagatzemi part o totes les dades que generi durant una simulació en un medi estructurat que permeti la conservació i posterior anàlisi de les dades. Així dons, l'eina de simulació implementa en el fitxer *dbManager.py* una interfície d'emmagatzemament en una base de dades MySQL.

Mitjançant la classe *DBManager* el simulador disposa de un conjunt de mètodes per anar emmagatzemant els "esdeveniments" que es generen durant una simulació (per exemple, quan es genera un missatge o una connexió). En particular es disposen de funcions per emmagatzemar la creació de missatges, nodes i connexions.

No obstant això, a dia d'avui, és possible que no existeixin totes les funcionalitats necessàries en el BTC-NS per assegurar la recopilació de totes les dades necessàries per l'estudi de l'eficàcia de l'aranya.

6.4 SimPy

Per executar les simulacions s'empra SimPy [20], un marc de treball per realitzar simulacions basades en processos i esdeveniments discrets. Per ajudar en la comprensió del que s'exposarà en els propers capítols, plantejarem a continuació una introducció als seus principals components: els esdeveniments discrets, el temporitzador i els processos.

Pel que fa als magatzems, ja els hem exposat anteriorment (en l'apartat 6.2.1.3), pel que no els tornarem a reproduir.

6.4.1 SimPy: esdeveniments discrets

L'esdeveniment discret de SimPy és, a grans trets, un semàfor que permet bloquejar l'execució del programa fins que l'esdeveniment es desbloqueja explícitament. Alternativament, en lloc de bloquejar el codi, pot executar una funció concreta (o *callback*) quan el semàfor s'activi.

El semàfor es pot crear mitjançant la invocació de la funció *event* de SimPy, tal com es mostra en la primera línia de codi, i pot invocar-se per bloquejar el codi tal com s'indica en la segona línia:

```
self.semaphores['dns_seeds'] = simpy.Environment.event()
yield self.semaphores['dns_seeds']
```

Codi 11: exemple d'esdeveniments discrets de SimPy

Per desbloquejar el semàfor cal invocar un de les següents funcions:

```
self.semaphores['dns_seeds'].succeed()
self.semaphores['dns_seeds'].fail()
```

Codi 12: exemple d'invocació d'un esdeveniment de SimPy

Les dues funcions desbloquejaran el semàfor, permetent que es continuï executant el codi, però de diferent manera: el primer indica que l'esdeveniment ha tingut èxit i el segon que ha fallat (on se li pot indicar el tipus d'error passant-li una excepció). Posteriorment es podrà comprovar per quin dels dos motius el semàfor s'ha alliberat.

6.4.2 SimPy: temporitzadors

Els temporitzador de SimPy és una especialització dels esdeveniments discrets, que permeten configurar un semàfor per tal que deixi de bloquejar el codi quan hagi transcorregut un temps determinat. La creació i invocació d'un temporitzador es realitza amb les següents línies de codi:

```
yield self.env.timeout(temps)
```

Codi 13: exemple d'invocació d'un temporitzador de Simpy.

6.4.3 SimPy: combinació de semàfors

Els diferents tipus de semàfors poden combinar-se per tal d'aturar l'execució del programa fins que es compleixin una o més condicions. Per combinar els semàfors es poden utilitzar les operacions booleanes '*or*' i '*and*' (que tenen un comportament similar al seu funcionament habitual).

```
# Wait for the answer
yield self.env.timeout(Node.DNS_TIMEOUT) |
      self.semaphores['dns_seeds']
```

Codi 14: exemple de combinació de semàfors de SimPy

Per exemple, en el cas anterior, el programa s'aturarà en aquests dos semàfors fins que un dels dos s'activi (és a dir, fins que un dels dos elements implicats torni l'equivalent de "*cert*", al igual que si es tractés d'una operació booleana convencional). Si hagés utilitzat el comparador "and" els dos semàfors s'haurien d'haver activat perquè el programa continués la seva execució.

6.4.4 SimPy: processos

Els processos de SimPy permeten executar en paral·lel[42] diversos mètodes i funcionalitats del simulador. Per exemple, en el següent codi (que correspon a la funció *Node.start*) s'invoquen tres processos, que s'executaran en paral·lel, i que donaran servei a les funcionalitats d'un node.

```
def start(self, network_discovery=False):
    """
    Start the node.

    :param network_discovery: Set to True if the node will work
    in a fully dynamic network.
    :type network_discovery: bool
    """
    # Receive network events
    self.env.process(self.receive_messages())

    # Dummy task to test simulation: send a network event every
    # t seconds (t in (1,10))
    self.env.process(self.generate_dummy_messages())

    if network_discovery:
        # Periodic network management
        if self.is_bitcoin:
            # Only for bitcoin's nodes
            self.env.process(self.periodic_network())
```

Codi 15: inici d'un node on s'executen diversos processos.

Els processos s'invoquen mitjançant el mètode *process*, que se li passa com a paràmetre la funció que executarà com un procés.

6.5 Conclusions

En els apartats anteriors hem enumerat (sempre emmarcant-nos en els aspectes relacionats amb la xarxa) els diversos tipus de simulació, la gestió de les connexions, l'agenda, el sistema de missatgeria i la base de dades. En aquest apartat, per concloure, intentarem donar una visió global de l'estat del simulador.

En particular, i atenent els diversos modes de funcionament del BTC-NS, en la següent taula podrem trobar un resum de les característiques del simulador d'acord al tipus de xarxa configurada.

[42] L'execució en paral·lel no té perquè ser una execució concurrent en diversos fils.

Id.	Aplicabilitat	Descripció	Implementat
NS-01	Xarxa estàtica	La gestió de les connexions la controla el simulador	Sí
NS-02	Xarxa dinàmica		Sí
NS-03	Xarxa estàtica	Els missatges es transmeten mitjançant l'identificador del node	Sí
NS-04	Xarxa dinàmica		Sí
NS-05	Xarxa estàtica	Els nodes no necessiten l'agenda d'adreces (no es descobreixen veïns ni hi ha noves connexions), ni els nodes DNS.	Sí
NS-06	Xarxa dinàmica		Sí
NS-07	Xarxa estàtica	Els simulador disposa d'un mecanisme per crear els nodes	Sí
NS-08	Xarxa dinàmica		Sí
NS-09	Descobriment de veïns	Els nodes implementen un mecanisme de descobriment de veïns	No
NS-10		Els nodes utilitzen les adreces de l'agenda.	No
NS-11		Els nodes implementen un mecanisme per inicialitzar la xarxa	No
NS-12		Els nodes implementen un mecanisme per connectar-se a altres iguals.	No
NS-13		Els nodes implementen un mecanisme per tancar connexions.	No
NS-14		Els nodes, com que poden tancar connexions (NS-13), necessiten un mecanisme per gestionar la caducitat dels missatges.	No
NS-15		Els nodes implementen un mecanisme per mantenir connexions	No
NS-16		Els nodes disposen d'un sistema d'adreces unificat.	No
NS-17		Els nodes segueixen el protocol de connexió de Bitcoin[43]	No
NS-18		L'aranya analitza periòdicament la xarxa.	No
NS-19		Els simulador disposa d'un mecanisme per crear els nodes.	No
NS-20		El simulador disposa d'un mecanisme per inicialitzar les dades del node DNS.	No
NS-21		L'agenda implementa un sistema estocàstic suficient per la gestió de les adreces.	No
NS-22	Comú	Existeix un mecanisme per transmetre un missatge a tots els nodes amb el que s'ha establert connexió.	No
NS-23		Els nodes utilitzen un únic sistema de transmissió de missatges.	No
NS-24		Els nodes consideren la latència de xarxa.	No

[43] Intercanvi de missatges *version* amb caràcter obligatori per establir la connexió.

Id.	Aplicabilitat	Descripció	Implementat
NS-25	Comú	S'emmagatzema la informació de l'aranya	No
NS-26		Simular el comportament de l'agenda si el mètode de funcionament del simulador no és "descobriment de veïns"	No
NS-27		No cal establir una connexió amb un node DNS per obtenir informació	No

Taula 19: resum de les característiques implementades en el BTC-NS.

Així mateix, també hem detectat un conjunt de tasques menors que s'han de dur a terme:

Id.	Aplicabilitat	Descripció	Resolt
NS-T1	Comú	El node *crawler* utilitza el nom de Bitnodes	No
NS-T2		En el gestor d'adreces s'emmagatzema un port, però no és necessari	No
NS-T3		En l'agenda d'adreces existeixen les variables *addr_know* i *_addr_to_send* que no s'empren.	No

Taula 20: llista de tasques a realitzar en el BTC-NS.

Amb l'anterior, podem procedir al següent capítol: planificar les tasques que s'han de dur a terme per tal que el simulador es comporti d'una forma similar al món real.

7 Millores en el simulador

En base al treball realitzat en els capítols 5 i 6, on analitzàvem, respectivament, el funcionament de la xarxa Bitcoin i del simulador BTC-NS; a continuació procedirem a detallar les modificacions que cal dur a terme en el simulador per tal d'aproximar el seu funcionament a la realitat pel que fa el descobriment de veïns.

Per exposar els canvis que s'han dut a terme en primer lloc presentarem la metodologia emprada com a base, tot seguit desgranarem les millores necessàries i, per finalitzar, detallarem, per cada millora, el problema, l'analitzarem i descriurem la solució implementada.

7.1 Metodologia de treball

Tal com indicàvem, prèviament a implementar els canvis que siguin oportuns, caldrà marcar un conjunt de paràmetres que serveixin de guia per dur-los a terme. En concret s'haurà d'establir el mecanisme de conservació del codi, seguir un estil de programació concret i determinar la metodologia de les proves unitàries.

Pel que fa la conservació del codi, s'emprarà el repositori subversion on està allotjat el simulador.

En relació a les proves unitàries, s'intentarà encapsular en mètodes les funcionalitats que s'implementin de tal forma que es facilitin les proves unitàries, que es realitzaran amb la utilitat unittest [21]. A més a més, aquells aspectes que no es puguin provar mitjançant proves unitàries es comprovaran amb les eines de depuració de codi de l'entorn de desenvolupament integrat (PyCharm[44]).

Per últim, pel que respecta a l'estil de programació, es procurarà seguir l'estàndard de Python establert en el PEP008 [22].

7.2 Millores necessàries

Si relacionem el funcionament de Bitcoin Core exposat en el capítol 5.7 i les característiques del simulador detallades en la secció 6.5, podem detectar un conjunt de modificacions que cal implementar en el simulador i que exposarem a continuació, separant tres petites modificacions (Taula 21) dels canvis més rellevants (Taula 22).

Les petites modificacions a realitzar són:

Id.	Origen	Depèn	Descripció
TK-01	NS-T1	-	Modificar el nom de la classe *BitnodesCrawler* a *Crawler*.
TK-02	NS-T2	-	Eliminar la variable *port* del gestor d'agendes.
TK-03	NS-T3	-	Eliminar les variables *addr_know* i *_addr_to_send* de l'agenda.

Taula 21: llista de modificacions simples a implementar en el simulador

I, pel que respecta a les modificacions més complexes, són:

Id.	Origen	Depèn	Descripció
TK-04	NS-24	-	Implementar la latència de xarxa en el sistema de missatgeria el simulador.

[44] https://www.jetbrains.com/pycharm/ [data de consulta 2015-01-03]

Id.	Origen	Depèn	Descripció
TK-05	NS-25	TK-11	Emmagatzemar la informació necessària de l'aranya.
TK-06	NS-23	-	Dissenyar i implementar un sistema de missatgeria únic.
TK-07	NS-22	TK-06	Establir un mecanisme per transmetre un missatge entre tots els nodes amb els que s'ha establert comunicació.
TK-08	NS-21	TK-13	Analitzar les característiques estocàstiques de l'agenda del simulador.
TK-09	NS-20	-	Implementar un mecanisme per inicialitzar les dades del node DNS.
TK-10	NS-19	TK-13	Implementar un mecanisme per crear nodes en el mode "descobriment de veïns".
TK-11	NS-18	TK-06 TK-10	Establir un mecanisme per tal que l'aranya analitzi periòdicament la xarxa.
TK-12	NS-17 NS-12	TK-06	Dissenyar el mecanisme d'establiment de connexió entre els nodes en el mètode de "descobriment de veïns".
TK-13	NS-16	-	Establir un mecanisme d'adreces concret.
TK-14	NS-15 NS-14 NS-13	TK-06	Implementar un mecanisme per tal que els nodes puguin mantenir les connexions.
TK-15	NS-11	TK-06 TK-09	Establir un mecanisme per tal que els nodes puguin inicialitzar la xarxa.
TK-16	NS-10	TK-12 TK-13 TK-06 TK-13	Implementar l'ús de l'agenda per part dels nodes.
TK-17	NS-09	TK-10 TK-12	Implementar el mecanisme de descobriment de veïns.
TK-18	NS-26	TK-16	Implementar un mecanisme per simular l'ús de l'agenda.
TK-19	NS-27	-	Els nodes DNS no funcionen de forma externa al mecanisme de connexions de Bitcoin.

Taula 22: llista de les modificacions complexes a implementar en el simulador.

Per cada una de les modificacions anteriors, exposarem a continuació la seva motivació (el "problema"), els canvis proposats ("l'anàlisi") i finalment la solució implementada. Com que alguns dels elements anteriors estan fortament acoblats, i és complicat separar-los, en ocasions proporcionarem la solució implementada de forma conjunta, agrupant diversos elements.

7.2.1 TK-04: latència en el simulador

Problema

La latència en la transmissió de missatges està dissenyada per simular el temps que tarda un missatge en transmetre's entre dos iguals. Per implementar-la, es considera el temps teòric que hauria de tardar un missatge d'acord a la seva mida, la latència i l'ample de banda de la connexió entre els dos iguals. El resultat d'aquesta operació s'aplica a un temporitzador (que enviarà el missatge quan s'esgoti el temps calculat).

A dia d'avui, el simulador disposa de diverses mètodes per obtenir l'ample de banda i la latència entre dos nodes[45], no obstant aquests valors no s'apliquen en la transmissió de

[45] En l'apartat 9.2 exposem diverses millores que es poden introduir en el sistema.

missatges. D'altre banda, pel que respecta a la mida dels missatges, no es disposa d'una funcionalitat per obtenir-la en tots els casos.

Anàlisi
La mida del missatge s'implementa parcialment en un mètode denominat *size* de la classe *Message*, que calcula la mida com la suma de l'espai en memòria dels tres atributs de la classe. Per tant caldrà estendre aquesta funcionalitat a tots els missatges del simulador, el que es podrà realitzar implementant la funcionalitat en cada missatge o emprant herència (tots els missatges implementats deriven de la classe *Message*).

En relació a l'ample de banda i la latència que es troben emmagatzemats en la classe *Link*, caldrà implementar una funció que, rebent com a paràmetre la mida del missatge, retorni el temps que tardaria en transmetre's mitjançant una simple operació:

$$Temps\ de\ transmissió = \frac{tamany\ missatge}{ample\ de\ banda} + latència$$

Per últim, caldrà combinar les dues funcions anteriors per tal d'enviar el missatge amb el retard oportú.

En resum es proposen les següents modificacions:

A. Implementar en tots els missatges un mètode *size* que permeti obtenir la mida.
B. Implementar en la classe *Link* un mètode per calcular la latència.
C. Invocar les dues funcionalitats anteriors en els mètodes de *Socket* des d'on s'envien els missatges.

Solució implementada
Pel càlcul de la mida dels missatges s'ha implementat un mètode en la classe *Message*, que per herència, s'implementa en tots els tipus de missatges. Inicialment es va implementar el següent codi[46], que empra les funcionalitats de reflexió de Python per obtenir dinàmicament totes les variables de cada missatge i calcular-ne la mida.

```
def size(self):
    return sum([getsizeof(getattr(self, attr))
                for attr in vars(self) if not attr.startswith("__")])
```

Codi 16: càlcul de la mida d'un missatge.

I d'altre banda, s'ha implementat una funció en la classe *Link* que calcula el temps de transmissió:

```
def msg_delay(self, msg_size):
    # Test case: test_network.test_6_network_latency
    try:
        return (msg_size / float(self.bandwidth)) + self.latency
    except ZeroDivisionError:
        return self.latency
```

Codi 17: càlcul del temps de transmissió

Per últim en la classe *Socket* s'han combinat els dos elements anteriors.

[46] En el capítol 8.3.2 exposem un canvi en el mètode de càlcul de la mida del missatge. La implementació aquí detallada funciona correctament amb missatges "petits", però no quan els missatges contenen moltes dades. En aquests casos el resultat de *size* és massa elevat i provoca que el seu retard sigui superior a les 8.000 unitats de temps del simulador.

7.2.2 TK-05: emmagatzemar la informació de l'aranya

Problema
Per dur a terme l'estudi objecte del projecte, cal assegurar que totes les dades necessàries s'emmagatzemen de forma estructurada en un medi persistent. Per determinar si disposarem de totes les dades hem de considerar els següents aspectes:

- Els nodes podrien arribar a connectar-se o desconnectar-se de la xarxa, pel que no és suficient emmagatzemar totes les adreces que l'aranya coneix al final de la simulació.
- L'aranya recopilarà un alt volum d'adreces, pel que no és factible llençar una crida a la base de dades cada cop que se'n rep una.
- L'aranya pot rebre una adreça diversos cops, pel que seria interessant conèixer quan es reben (i, per tant, poder emmagatzemar diversos cops la mateixa adreça).
- L'aranya establirà connexions amb diversos nodes de la xarxa per tal de trametre missatges sol·licitant adreces d'altres nodes.
- Cal controlar tots els vectors d'entrada de les adreces als diferents nodes de la simulació, incloent les adreces proporcionades a l'arrancar l'aranya.

Anàlisi
I en base a les característiques anteriors podríem dur a terme tres aproximacions:

A. Implementar una memòria intermediària en la classe *DBManager*. D'aquesta forma evitaríem fer una crida en la base de dades cada cop que volguéssim salvar un element i l'invocaríem quan tinguéssim N elements del mateix tipus [23].
B. Si emmagatzemem un missatge *addr* amb totes les adreces que conté, únicament hauríem d'emmagatzemar el moment en que l'aranya rep dits missatges. Amb aquesta referència podríem conèixer les adreces rebudes.
C. Guardar tota la informació quan finalitzi la simulació. Aquesta opció implicarà que necessitarem més memòria per emmagatzemar tota la informació.

A banda de l'exposat anteriorment, també caldrà assegurar que la base de dades guardi les adreces incorporades als nodes obtingudes mitjançant servidors DNS o a partir de les adreces codificades en el propi programari.

Solució
De les tres opcions anteriors, deixarem l'opció A com una possible millora en el simulador (exposada en l'apartat 9.4), descartarem l'opció C degut a la seva complexitat i optarem per la solució B.

Conseqüentment, s'ha modificat l'emmagatzemament dels missatges *addr* per tal que guardin totes les adreces que reben. Atenent que un missatge *addr* pot contenir fins a mil adreces, en lloc d'emmagatzemar aquestes adreces en una taula específica, s'ha decidit guardar-les en una variable de tipus *CLOB*.

En aquest sentit, per emmagatzemar-les en un *CLOB* s'ha optat per emprar el següent mecanisme de serialització, on les adreces es separen amb el caràcter '|'.

origin	destination	addr_dict
50593841	51052585	50593841
50462752	51249166	51183659|51249166
51249181	50462722	50397185|50462722|50659348|50331693

Imatge 23: exemple de l'emmagatzemament de les adreces

Així mateix s'ha afegit a la base de dades la taula *EvHardcodedBootstrap* que s'encarrega d'emmagatzemar les adreces que un node incorpora a partir de les adreces codificades en el programa.

7.2.3 TK-06, TK-07: sistema de missatgeria únic.

Problema
Tal com detallàvem en l'apartat 6.2.1.3, el simulador disposa de dos sistemes de missatgeria: d'una banda la funció *send_message* i *send_broadcast_message* i, d'altra banda, la funció *_send_messages_to_all* (invocada des del mètode *periodic_network*, que sembla encarregar-se de realitzar les connexions de xarxa).

Per tant, disposem de dos sistemes per trametre missatges que s'haurien d'unificar en un sol sistema per evitar la complexitat en el codi.

Anàlisi
Pel que respecta les funcions *Socket.send_message* i *Socket.send_broadcast_message* trameten un missatge al igual on va dirigit quan el reben (amb l'aplicació de la latència i ample de banda comentades anteriorment.

D'altre banda, la funció *_send_mssages_to_all*, du a terme els següents passos:

1) Invoca a *self.addrmgr.get_random()*, que retorna una única adreça aleatòria del node.
2) Recorre totes les adreces de l'agenda.
3) Quan troba l'entrada a l'agenda seleccionada en el primer punt invoca la funció *Node.address_to_send*, que tornarà les adreces "pendents d'enviar" ubicades a *Node.NetAddressInfo._addr_to_send*.
4) Si el llistat de direccions del punt 3) no està buit, s'enviarà al node seleccionat en el punt 1) un missatge amb totes les adreces de 3) mitjançant *Socket*.*send_message*.

Respecte als dos mètodes anteriors, podem veure com s'han implementat dues aproximacions diferents per la tramesa de missatges en la xarxa: en primer lloc tenim la tramesa directa i "immediata" (implementada per *Socket.send_message*) i, en segon lloc un mètode asíncron inspirat en Bitcoin Core (del qual proporcionem més informació en el tercer annex) però que no està completament finalitzat.

Solució
Com que el simulador no és un sistema distribuït i asíncron (tot i que l'emula) considerem que no és necessari arribar a implementar l'arquitectura per la tramesa de missatges de Bitcoin Core, ja que no podem apreciar els avantatges que compensin la dificultat derivada de la nova arquitectura.

En aquest sentit considerem que la millor opció és emprar el sistema de missatgeria implementat per les funcions *Socket.send_message* i *Socket.send_broadcast_message*, eliminar la funció *_send_messages_to_all* i utilitzar la funció *periòdic_network* (detallarem

més endavant com) per realitzar les funcions periòdiques de la xarxa implementades en *_send_messages_to_all .*

I en conseqüència, s'ha procedit a eliminar la funció *_send_messages_to_all* i definir *Socket.send_message* i *Socket.send_broadcast_message* com els mètodes responsables, respectivament, per la tramesa de missatges a un sol node i a tots els nodes amb el que es manté una connexió.

7.2.4 TK-08: anàlisi de les característiques estocàstiques

Problema
Per assegurar que el simulador tingui un comportament fidedigne respecte al client de referència, caldrà analitzar les característiques estocàstiques del gestor d'adreces.

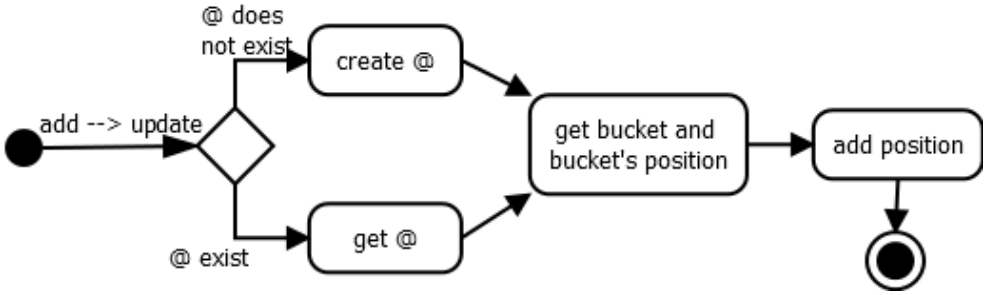
Anàlisi
La interacció d'un node amb la seva agenda, tal com hem exposat en apartats anteriors, succeeixen en cinc moments:

I. Addició d'una adreça al gestor (funció *add*).
II. Elecció d'una adreça per realitzar una nova connexió (funció *select*).
III. Elecció de diverses adreces per respondre un missatge *getaddr*.
IV. Marcar una adreça com intentada.
V. Marcar una adreça com connectada.

Tot seguit, per cada un dels punts procedirem a analitzar el funcionament del simulador.

7.2.4.1 Addició d'una adreça al gestor

Anàlisi
En el simulador, quan s'ha d'afegir una nova adreça a l'agenda cal invocar la funció *NetAddressManager.add*. Aquesta funció segueix el següent procediment:



Imatge 24: procediment en el BTC-NS per afegir una nova adreça.

En el procediment, abans del pas "obtenir sac i posició" s'escollirà si s'actua sobre el grup de les adreces "provades" o les "noves". Per defecte sempre s'actuarà sobre el sac de les adreces "noves" amb dues excepcions: que el node del que hem obtingut l'adreça sigui el mateix que l'adreça a introduir o que l'adreça ja estigui en les adreces "provades".

En cas que la posició en les adreces "noves" ja estigui ocupada, s'eliminarà l'adreça antiga, independentment de si és, o no, "millor". De forma, similar, la posició en les adreces "provades" està ocupada, independentment de quina adreça sigui "millor", es mourà l'adreça que ocupava la posició al sac de les "noves".

El comportament és similar al de Bitcoin Core exceptuant que no s'actualitzen les dades d'una adreça mai (al client de referència la direcció s'actualitza les adreces d'acord a una probabilitat).

Solució
Atenent que en el desenvolupament del gestor d'adreces es va tenir en compte aquest fet, que existeixen comentaris indicant que és una elecció en el desenvolupament i que no afecta a les condicions estocàstiques del gestor, no es realitzarà cap modificació en el seu comportament

7.2.4.2 Elecció d'una adreça per realitzar una connexió

Anàlisi
El procediment implementat per aquesta funcionalitat és pràcticament idèntic al detallat pel client de referència en l'apartat 5.5.4.1 amb una única diferència: en lloc d'emprar els mecanismes concrets d'aleatorietat implementats en Bitcoin Core, es recolza en bona part amb la biblioteca *numpy* per l'elecció d'elements aleatoris[47].

Solució
Conseqüentment es considera que el funcionament és pràcticament idèntic, pel que no cal dur a terme cap tasca.

7.2.4.3 Elecció de diverses adreces per respondre un missatge *GetAddr*

Anàlisi
El procediment per obtenir múltiples adreces es realitza mitjançant la invocació de la funció *NetAddressManager.get_addressk* i, si el comparem amb el client de referència, és força similar al descrit en la Imatge 16 (pàgina 43), pel que únicament exposarem les diferències:

- Ús de la variable *vRandom*. Tal com hem vist en l'apartat 5.5.4.2, Bitcoin Core empra un vector, denominat *vRandom*, que conté totes les adreces desordenades. En l'altre extrem, el BTC-NS, no disposa d'aquesta variable, però empra la biblioteca *numpy* per dur a terme una selecció aleatòria[46].

- Cada cop que el client de referència selecciona una adreça del vector *vRandom*, el desordena parcialment intercanviant les posicions dels seus membres. Aquest comportament no es reprodueix en el simulador.

Solució
Tot i que l'aproximació de les dues implementacions és diferent, no s'ha realitzat cap canvi al existir una selecció aleatòria d'adreces similar[48].

7.2.4.4 Marcar una adreça com "intentada"

Anàlisi
Per tal de marcar una adreça com a "intentada" en el simulador cal invocar la funció *NetAddressInfo.attempted*, que augmenta en u el comptador d'intents, un funcionament anàleg al client de referència.

Solució
S'ha afegit la crida corresponent al simulador.

[47] http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.choose.html [data de consulta 2015-01-04].
[48] Cal matisar que no s'ha realitzat un estudi exhaustiu dels dos mètodes.

7.2.4.5 Marcar una adreça com a connectada

Anàlisi
Per marcar una adreça com a connectada cal invocar l'adreça *NetAddressManager*.connected, que té un funcionament similar al client de referència i que es pot veure en la Imatge 14 de la pàgina 41.

Solució
S'ha afegit la crida corresponent al simulador.

7.2.5 TK-09: mecanisme per inicialitzar els nodes DNS

Problema
Actualment els nodes que representen un servidor *DNS* no s'inicialitzen, pel que no poden servir peticions d'altres nodes en el simulador.

Anàlisi
Aquest tipus de nodes té implementat un sistema d'inicialització per tal que, en el moment de la construcció de l'objecte, rebi una llista de "llavors" (traducció literal de l'anglès *seeds*) que seran les adreces que proporcionarà el servidor quan se li realitzi una consulta.

Per tal d'emprar aquest mecanisme (implementat pel patró constructor *NodeProfiles.builder* i la classe de configuració *NodeProfilesConfigDNS*) és necessari que en el moment de la construcció d'aquest tipus de node es conegui les adreces dels altres nodes de la xarxa per tal d'incloure'ls en el constructor.

No obstant, en el sistema actual d'inicialització del simulador, tots els nodes es construeixen alhora, pel que no es poden obtenir les adreces IP abans d'inicialitzar els objectes i, en conseqüència, no es podrà passar una llista d'adreces al patró constructor que generarà els nodes DNS.

Tot i això, cal considerar que el simulador disposa de dos modes de funcionament: el normal (on es configuren alguns valors de la simulació i es calculen o generen aleatòriament la resta) i l'avançat (on es defineix detalladament la xarxa que es simularà). Per tant, caldrà disposar d'un mecanisme que permeti inicialitzar els servidors en el mode "normal" i el mecanisme ja existent pel mode avançat.

En un altre ordre de coses, ens hem de plantejar la necessitat del node DNS en tots els mètodes de funcionament del simulador. D'acord al exposat en l'apartat 6.1, el simulador disposa de tres mètodes de funcionament principals: "xarxa estàtica", "xarxa dinàmica definida" i "xarxa dinàmica". En aquest context els nodes DNS únicament tenen sentit en el tercer mètode, ja que en els dos primers no cal arrancar la xarxa –la raó d'existència del node DNS.

En conclusió, cal desenvolupar un mecanisme que permeti als DNS disposar d'un conjunt d'adreces (entre tots els nodes convencionals de la xarxa) que els permeti servir les seves peticions d'acord als següents punts:

A. Abans de l'inici de la simulació, obtindré el número de nodes, preveure quines adreces de xarxa tindran i establir que un tant per cent d'aquestes adreces alimentin el simulador.
B. El tant per cent d'adreces s'haurà de codificar de tal forma que, en un futur, pugui ser un dels paràmetres de configuració de les simulacions.

C. Addicionalment es podria afegir un paràmetre addicional per afegir un tant per cent d'adreces invàlides: adreces de xarxa que mai correspondran a un igual actiu.

Addicionalment, es proposa com a millores futures al simulador els següents punts:

- Limitar el funcionament del servidor DNS a la "xarxa dinàmica" (apartat 9.8).
- Disposar d'un mecanisme que permeti establir el nivell de servei dels nodes. En aquest sentit, el nivell de servei seria directament proporcional a la probabilitat que l'adreça d'un node formi part del servidor DNS (apartat 9.7).

Solució
Per inicialitzar els servidors DNS s'ha procedit a implementar les següents funcionalitats en la classe *Network* (fitxer *network.py*):

Primer de tot calia determinar el número d'adreces de nodes que proporcionaran els servidors DNS. Per calcular aquesta xifra s'ha definit la constant estàtica *Network.PERCENT_NODES_DNS*[49], que permet obtenir el número màxim de nodes a servir amb un simple percentatge entre el número de nodes de la simulació i la variable.

En segon lloc, per obtenir les adreces dels iguals que proporcionarà cada servidor, per cada node DNS es recorren tots els nodes del sistema escollint aleatòriament les adreces tal com es mostra en el següent codi:

```
# Hardcoded DNS bootstrapping
for n in self.nodes:
    if n.is_dns:
        # All nodes will have the same DNS seeds
        Node.dns_seeds.add(n.get_address)
        # Now we'll initialize the DNS itself
        debug_nodes_per_dns[n.id] = set()
        dns_nodes.append(n)

        for i in range(num_nodes_per_dns):
            peer = self.nodes[numpy.random.randint(0,
                              self.num_nodes-1)]
            if peer.is_bitcoin:
                n = n  # Cheating the IDE
                """:type:node.DNSSeed"""
                n.addresses.add(peer.get_address)
                debug_nodes_per_dns[n.id].add(peer.get_address)
```

Codi 18: generació d'adreces per cada DNS (*Network. create_network_discovery*)

En aquest sentit, per evitar que els nodes DNS serveixin adreces de nodes que no formin part de la xarxa Bitcoin (com servidors DNS o aranyes) s'ha implementat en la classe *Node* la funció *is_bitcoin*, que torna *cert* si el node forma part de la xarxa Bitcoin i fals en

[49] La constant s'ha definit en 25, pel que els servidors DNS disposaran com a màxim el 25% de les adreces del simulador. L'elecció d'aquest valor ha estat subjectiva i es proposa com a futura millora que sigui proporcional al número de nodes de la simulació (apartat 9.8).

cas contrari[50]. Aquesta funció està implementada en la classe *Node* del simulador pel que, per herència, es troba implementada en tots els nodes.

Així mateix, s'han implementat dues funcions més *Node.is_crawler* i *Node.is_dns* que tornen cert si els nodes són una instància de les classes esmentades.

```
@property
def is_crawler(self):
    """ :return: True if it's a crawler node."""
    return isinstance(self, Crawler)

@property
def is_bitcoin(self):
    """ :return: True if it's a Bitcoin's node (i.e. works
        over the Bitcoin's network)."""
    return not(self.is_dns or self.is_crawler)
```

Codi 19: implementació de la funció *Node.is_bitcoin* (*Node*).

Per últim, com a decisió de disseny, hem establert que tots els nodes tinguin codificats els mateixos servidors DNS emprant la variable estàtica *Node.dns_seeds*.

7.2.6 TK-10: creació d'iguals en el mode "descobriment de veïns"

Problema
En el simulador no està desenvolupat el mode "descobriment de veïns" que implementa una "xarxa dinàmica".

Anàlisi
Pel que fa a les opcions "xarxa estàtica" i "xarxa dinàmica definida" del simulador, els iguals de la xarxa es creen respectivament en les funcions *Network.create_erdos_model* i *Network.create_barbasi_albert_model*. No obstant, per la "xarxa dinàmica" no existeix un mètode responsable de la creació dels nodes o de la inicialització de la xarxa.

Així dons, caldria implementar una funció que permeti crear i inicialitzar els nodes en la xarxa dinàmica. El mètode es responsabilitzaria de crear tots els nodes (segons les indicacions de l'usuari) i d'inicialitzar-ne les funcions de xarxa dels nodes.

Solució
Per tal de crear i inicialitzar els nodes de la xarxa en el mode del simulador *xarxa dinàmica*, seguint l'exemple dels altres modes del simulador, s'ha implementat la funció *Network. create_network_discovery* que realitza les següents tasques:

A. Crea els nodes d'acord a la configuració proporcionada per l'usuari.
B. Realitza les inicialitzacions prèvies de la xarxa com, per exemple, la inicialització dels servidors DNS exposats en l'apartat 7.2.4.
C. Inicialitza la xarxa de cada node mitjançant la invocació de la funció *Node.start*. Aquesta funció, a la seva vegada, inicialitza els diversos processos periòdics que cada node durà a terme cada node durant la simulació.

[50] En el món real, l'aranya formaria part de la xarxa Bitcoin ja que es comunica amb el seu protocol, no obstant, a efectes pràctics se l'exclourà ja que creiem que en el món real no es consideraria un "node" de confiança al estar les seves funcionalitats limitades a l'anàlisi.

7.2.7 TK-11: anàlisi periòdic de la xarxa per l'aranya

Problema
Per tal de dur a terme el projecte, l'aranya ha d'analitzar contínuament la xarxa procurant detectar el màxim número possible de nodes.

Anàlisi
El node tipus aranya disposa de dos mètodes: *_receive_message_addr* i *_send_discovery_msg*. El primer espera rebre un missatge *addr* i, davant d'aquest missatge n'analitza les adreces i, quan troba alguna adreça que no coneix, li envia un missatge *GetAddr* per preguntar-li les adreces que coneix el nou node.

Aquest mecanisme presenta dos inconvenients: si, per qualsevol motiu un node no contesta a un missatge *GetAddr*, llavors l'aranya aturarà el seu funcionament (el que no és acceptable). D'altre banda, el node prescindeix del mecanisme per establir les connexions de Bitcoin, pel que qualsevol missatge que enviï serà rebutjat.

Conseqüentment es proposa el següent:

A. En la construcció de l'objecte aranya s'ha d'activar un procés que cada *x* segons intenti realitzar noves connexions per trametre missatges *GetAddr*.
B. L'aranya rebutjarà peticions de connexió d'entrada.
C. Si l'aranya reutilitza atributs de *Node*, obtinguts per herència, s'haurà d'assegurar que no es trameten missatges que no siguin necessaris.
D. Es classificaran i emmagatzemaran les adreces d'una forma diferent a la dels nodes convencionals.

Solució
Per dur a terme les tasques abans mencionades, en primer lloc es va decidir limitar el número d'aranyes a un únic node per tal de facilitar la implementació del projecte i detectar els possibles errors que poguessin sorgir. En aquest sentit, s'ha modificat el mètode de creació dels nodes per tal de crear una única aranya, independentment del percentatge indicat.

Això comporta que, tal com es veurà a continuació, es pugui crear una aranya tot i que el percentatge d'aquell tipus de node sigui zero.

```
node_counter = 0
has_crawler = False
for k in profiles:
    # Except for the crawler... for now we will create a single
    # crawler.
    if k is NodeProfiles.CRAWLER and not has_crawler:
        has_crawler = True
        -_counter += 1
        self._nodes_profiles.append(k)

    limit = (self.num_nodes * profiles[k])//100
    for i in range(limit):
        self._nodes_profiles.append(k)
        node_counter += 1
```

Codi 20: creació del node aranya (*Network. create_network_from_models*).

En segon lloc, com que el tipus de node aranya deriva per herència de *Node*, s'han bloquejat tots els intents de connexió dels nodes mitjançant la implementació de la funció *Node. _process_received_message*, on només acceptem missatges *addr*, *version* i *reject*.

Així mateix, només acceptem aquells missatges *version* que són una resposta (és a dir, no admetem connexions d'entrada):

```
# We only accept version and addr messages
if not isinstance(msg, ds.MessageAddr) \
        and not isinstance(msg, ds.MessageVersion) \
        and not isinstance(msg, ds.MessageReject):
    return
[…]
# We only accept response message versions (no
# incoming connections for the crawler)
if isinstance(msg, ds.MessageVersion):
    if not msg.response:
        […]
        return
    elif not self.addrmgr.is_ongoing(msg.origin):
        […]
        return
```

Codi 21: limitació de les connexions de l'aranya en el simulador.

En tercer lloc, de forma similar a l'anterior, s'ha modificat el mètode *Crawler.send_message*, per tal que només s'enviïn els següents missatges: *version, verack, reject* i *getaddr*. Com que l'aranya implementa per herència les funcionalitats de Node, així ens assegurem que no enviï missatges que no li pertoquen[51].

Per últim, pel que fa a l'emmagatzemament de les adreces de l'aranya, com que empra bona part de les funcionalitats de *Node*, hem decidit implementar un sistema per la gestió de les adreces (denominat *NetAddressManagerCrawler*) similar al que implementen els nodes convencionals (denominat *NetAddressManager*).

L'estructura de dades implementada en el gestor de les adreces contempla el següent:

```
# * self._new: the addresses haven't been tried.
# * self._ongoing: we're trying a connection to the nodes stored in
#   this list. A connection has been successfully established when the
#   addr message has been received from the peer.
# * self._tried: the crawler has received addr mgs from these
#   addresses.
# * self._bad: the connection with these peers had timed out or we had
#   not received the addr message.
self._addresses = dict()  # Key => address
""":type: dict[int, NetAddressInfoCrawler]"""
self._new = list()
""":type:list[NetAddressInfoCrawler]"""
self._ongoing = list()
""":type:list[NetAddressInfoCrawler]"""
self._tried = list()
""":type:list[NetAddressInfoCrawler]"""
self._bad = list()
""":type:list[NetAddressInfoCrawler]"""
```

Codi 22: estructura de dades del gestor d'adreces del node

[51] Tal com hem comentat en alguna altre ocasió, en l'apartat 9.1, proposem un altre mecanisme per construir els nodes que pugui combatre aquest problema.

Així mateix, com que l'aranya utilitzarà bona part de les funcionalitats de *Node* serà necessari que el gestor d'adreces implementi els mateixos mètodes (*select*, *tried, add* i *good*) per assegurar-ne la compatibilitat. No obstant, la funcionalitat dels mètodes divergeix força de l'exposada anteriorment per Bitcoin Core ja que no necessitarem les funcionalitats estocàstiques del gestor d'adreça dels nodes convencionals.

En concret, la funcionalitat de *select*, que escollirà el següent node al que es connectarà l'aranya és el que es mostra a continuació:

```
def select(self, now):
    if self._new:
        target = self._new
    elif self._bad:
        target = self._bad
    elif self._tried:
        target = self._tried
    else:
        target = None

    if target:
        address = numpy.random.choice(target)
        """:type:NetAddressInfoCrawler"""
        max_attempt = max(address.last_attempted,
                          address.last_success)
        if address.last_attempted == 0 or \
           (max_attempt + self.TIME_TRYING_ADDRESS) < now:
            return address.address
    return None
```

Codi 23: comportament de la funció *select* de l'agenda de l'aranya

En el codi anterior podem veure com el node intentarà agafar una adreça d'acord a si s'ha provat o no, donant preferència a aquells nodes amb els que l'aranya no s'ha connectat. Addicionalment també s'eviten les connexions amb aquelles adreces amb les que fa poc temps que s'ha intentat una connexió.

La resta de mètodes, classificaran una adreça en una de les quatre llistes, actualitzant el número d'intents de connexió i el moment en que s'han realitzat.

7.2.8 TK-12: establiment de la connexió entre nodes.

Problema
En el mode "xarxa dinàmica", on el simulador no crea les connexions entre els diferents parells de la xarxa, no es troba implementat el mecanisme que permeti als nodes connectar-se entre sí d'acord a l'exposat en l'apartat 5.1.

Anàlisi
En l'inici del projecte, el mecanisme per establir una connexió entre nodes està parcialment definit en la funció *Node.periodic_network*. Aquest mètode s'executa de forma periòdica per tal d'establir noves connexions (sempre i quan no s'hagi arribat al màxim). Les funcionalitats codificades en el mètode tenen les següents mancances:

1) Cada cop que s'executa la funció, si no s'han establert totes les connexions possibles, s'invocarà la funció *_use_seeds* per tal d'afegir a l'agenda els nodes codificats directament en el programa. A més a més, el programari no empra els DNS per la inicialització de la xarxa d'acord a l'exposat en l'apartat 5.3.1.

2) Al crear una connexió entre dos nodes (representada per un objecte Link) no es considera l'ample de banda i la latència.
3) Es tramet el primer missatge *version*, però no s'espera a rebre el segon missatge *version* per considerar que la connexió està establerta. En altres paraules: abans d'enviar o rebre el segon missatge *version* els nodes es poden comunicar i, a més a més, es creen dos cops les connexions (en la funció *periòdic_network* i quan es rebi el segon missatge *version*).
4) La funció *_process_received_message*, no rebutja l'intent de connexió si els dos iguals implicats empren una versió del protocol Bitcoin incompatible. Aquesta característica no es podrà implementar en el marc del projecte i es deixa pendent com una millora futura (apartat 9.8).
5) No s'emmagatzema la versió del node amb el que establim una connexió.
6) No es marca l'adreça com a "provada" ni l'intent de connexió en l'agenda d'adreces.

Conseqüentment es proposen els següents canvis:

A. Establir un mecanisme per inicialitzar la xarxa dels nodes (tal com s'exposa en 7.2.11).
B. S'ha d'impedir enviar o rebre missatges si no hi ha una connexió oberta (a excepció dels missatges *version*, *verack* i tots els DNS). En aquest sentit, en futures versions del simulador caldria considerar la conveniència que els nodes *DNS* emprin el mateix sistema de connexions que la resta de nodes (millora en l'apartat 9.1).
C. No es diferenciaran les connexions d'entrada de les de sortida.
D. Caldrà crear l'objecte *Link* que representi cada connexió entre dos nodes. Per crear-lo caldrà obtenir uns valors de latència i ample de banda d'acord a la configuració indicada per l'usuari.
E. S'ha d'establir un mecanisme que permeti controlar l'estat d'una connexió d'acord al protocol d'intercanvi de missatges de Bitcoin. Addicionalment, caldrà controlar quan s'han enviat els missatges a fi i efecte d'establir la caducitat de les connexions. En aquest sentit, no cal controlar la condició de si es un missatge inicial o de resposta perquè el propi simulador ho controla amb la variable *MissageVersion.response*.
F. Emmagatzemar la versió del protocol Bitcoin en una connexió entre els dos nodes.
G. Implementar a la classe *Link* un atribut que ens permeti conèixer la versió de l'igual.

A més a més, per tal de simular el comportament dels nodes en la vida real, caldrà implementar un sistema per tal que aquests "es desconnectin" temporalment de la xarxa de forma aleatòria i durant un temps determinat.

Solució
Per tal de descriure la solució implementada, degut a la seva complexitat, la dividirem en diverses parts. Així mateix, també exposarem la solució al manteniment de les connexions per part dels nodes, problema que veurem en l'apartat 7.2.10.

En un altre ordre de coses, per tal de determinar quan un node està actiu o no, s'ha afegit un nou semàfor denominat *active* que ho controla. Un node que no estigui actiu no podrà enviar missatges, realitzar altres tasques o rebre missatge (qualsevol missatge rebut quan un node no estigui actiu es descartarà). A més a més, amb una P=0.5 el node cancel·larà silenciosament (és a dir sense avisar als seus iguals) totes les connexions que tingui obertes.

Per determinar quan un node deixa d'estar temporalment actiu s'ha implementat el següent codi en la funció *Node.periodic_network*:

```
# Determine if the node goes off-line
if self.semaphores['active']:
    random = numpy.random.randint(0, 9999)
    if random >= 9900:
        self.semaphores['active'] = False
        if db.active:
            db.manager.save_event_up_down(self.env.now, self.id,
                                          up=False)
        un_active_period = numpy.random.random()
        multiplier = float(numpy.random.randint(10, 100))
        self.log.debug("%7.4f: Node |%d| (IP |%d|) is down." %
                       (self.env.now, self.id, self.ip))
        if numpy.random.choice((True, False)):
            # With P=0.5 we drop all connections
            self.socket.clear_links()
            self.num_connections = 0

        yield self.env.timeout(un_active_period * multiplier)
        self.semaphores['active'] = True
        if db.active:
            db.manager.save_event_up_down(self.env.now, self.id,
                                          up=True)
        self.log.debug("%7.4f: Node |%d| (IP |%d|) is up." %
                       (self.env.now, self.id, self.ip))
# End off-line
```

Codi 24: activació o desactivació d'un node

Com es pot veure, la inactivitat del codi es computa amb la multiplicació de dos nombres aleatoris: el primer està en el rang [0, 1] i el segon en el rang [10, 100], pel que el temps màxim que un node estaria inactiu són 100 unitats de temps en el simulador.

7.2.8.1 Estats d'una connexió

Per conèixer l'estat d'una connexió, a la classe *Link* s'hi han afegit quatre camps per controlar, respectivament, la versió de la connexió (*version*), l'última comunicació amb l'igual (*update_time*) i controlar l'estat de la connexió (*status*):

- *version*, que emmagatzemarà la versió del protocol Bitcoin que s'utilitza en una connexió.
- *update_time* que controlarà quan va ser l'últim cop que es van rebre notícies de l'altre extrem (el que ens permetrà controlar, més endavant, les caducitat de les connexions).
- *forward_addresses* que, tal com veurem més endavant, s'utilitzarà en el descobriment de veïns.
- *status* determinarà l'estat d'una connexió. Els possibles estats d'una connexió estan definits en la classe *LinkStatus*, que reproduïm parcialment tot seguit:

```
"""
Possible connection status:
* NONE: no connection between nodes.
* VERSION: Node A sent a version message to B.
* ESTABLISHED: connection between node A and node B established, the
               version messages have been exchanged. Default
               value for the Eredös and Barbasi modes.
* DONE: connection between node A and node B established. The version
        and verak messages have been exchanged. Bitcoin's handshake
        complete.
* DNS: for outbounds DNS queries.
* DNS_INBOUND: for inbound DNS responses.
"""
```

Codi 25: definició dels estats d'una connexió (network.py, *LinkStatus*)

7.2.8.2 Manteniment de les connexions

Pel que respecta al manteniment de les connexions, en el mètode *Node.periodic_network* s'han desenvolupat quatre tasques: la inicialització de l'agenda dels nodes (tal com hem explicat anteriorment), l'execució de part de les activitats de descobriment de veïns (que veurem més endavant) i el manteniment i establiment de noves connexions (que descriurem a continuació).

En les tasques de manteniment de la connexió, es recorren totes les connexions (independentment del seu estat) per comprovar si estan caducades i, si ho estan, es tanquen (es canvia el seu estat a "NONE" i, en una iteració posterior, s'eliminen). Per determinar la caducitat de les connexions s'ha definit en la classe *Link* les constants oportunes[52]:

```
NONE_TIMEOUT = 300
""" In the network discovery, the timeout for ongoing connections. """
VERSION_TIMEOUT = 600
""" In the network discovery, the timeout for the Version message. """
ESTABLISHED_TIMEOUT = 600
""" In the network discovery, the timeout for Established connections.
```

Codi 26: termini de les caducitats de les connexions (*network.py*, *Link*)

En aquest sentit, cada cop que un client rep un missatge d'un igual amb el que està connectat s'emmagatzemarà el moment en que s'ha rebut a fi i efecte de la comptabilització de la caducitat abans descrita.

D'altre banda, també s'han implementat els missatges *ping* i *pong* de Bitcoin (implementats per la classe *MessagePing*) que són enviats des de la funció *periòdic_network* amb una freqüència determinada per la constant *Node. PING_ANNOUNCEMENT*.

Tal com el seu nom indica, la funcionalitat d'aquests missatges és determinar si l'igual al que van adreçats està actiu i la resposta a un missatge *ping* serà un missatge *pong*. En aquest sentit, un missatge *ping* únicament s'enviarà quan s'hagi rebut el missatge *pong* previ. Aquest comportament està regulat per la variable *Link.waiting_ping*.

[52] Els valors s'han determinat de forma que la majoria de missatges transmesos no superen aquests valors.

```
def __init__(self, destination, origin, timestamp, ping=True):
    Message.__init__(self, destination, origin, timestamp)
    self.ping = ping
```

Codi 27: implementació del missatge ping i pong en la mateixa classe.

7.2.8.3 Realització de noves connexions

Després de realitzar el manteniment de les connexions, el mètode *Node.periodic_network* intenta obrir noves connexions en el node. Per intentar obrir-les es tenen en consideració dos aspectes: que no s'hagi superat el número màxim de connexions establertes i que no s'intentin establir més de dues connexions "de sortida" alhora.

```
# Try to establish new connections
if self.num_connections < self.prof_parameters['maxconnections'] and \
        len(self.semaphores['outbound']) < max_outbound_connections:
```

Codi 28: condicions per intentar establir una nova connexió.

El número de connexions establertes es controla mitjançant la variable *self.num_connections*, que augmenta en u quan s'intenta establir una nova connexió, ja sigui el propi node que l'iniciï o un node extern. Per tant aquesta variable conté el número de connexions establertes i el número de connexions en procés d'establiment.

En relació al límit als intents de connexions en curs, aquesta funcionalitat es va introduir com una millora per tal d'assegurar que els nodes es poguessin interconnectar. Sense aquesta limitació els nodes eren incapaços d'establir noves connexions, ja que tots els clients de la simulació obrien el màxim número de connexions de sortida i, per tant, rebutjaven totes les d'entrada.

Per aquest control es va definir el semàfor *self.semaphores['outbound']*, que conté una llista de totes les connexions que s'estan intentant establir. Les connexions s'eliminen d'aquesta llista quan caduquen o quan s'estableixen.

Per últim, en l'establiment de la connexió, el mètode consulta a l'agenda d'adreces mitjançant el mètode *select*, que torna un candidat per la connexió. Amb aquesta adreça del node crea la connexió (un nou objecte *Link*) marcant l'estat com a "*VERSION*" (el que indica que s'ha enviat el primer missatge), s'alimenta el semàfor que controla el màxim de connexions i s'envia el primer missatge *version*.

Les funcionalitats descrites es troben en els mètodes *Node.periodic_network* i *Node.connect*.

7.2.8.4 Recepció del primer missatge *version*

Quan un igual rep el missatge *version* inicial, en primer lloc comprovarà si pot rebre el missatge: els clients que no estiguin connectats amb l'igual únicament poden rebre missatges *version*, *reject*, a banda de missatges *DNS*.

Tot seguit, el node comprovarà si es pot establir la connexió amb l'igual. En concret comprovarà si ha arribat al número màxim de connexions i si no hi ha establerta una connexió. En cas que una de les dues connexions no es compleixi, s'enviarà un missatge *Reject* informant que la connexió no s'ha pogut establir. Si les dues condicions s'estableixen enviarà el segon missatge *version* a l'altre node i canviarà l'estat de la seva connexió amb l'igual a "*ESTABLISHED*".

Addicionalment el segon node obtindrà quina versió del protocol Bitcoin s'emprarà en la connexió, comparant la versió que figura en el missatge amb la seva pròpia versió i seleccionant la versió més petita. A dia d'avui, considerem que totes les versions són compatibles.

A partir d'aquest moment, la connexió es considera establerta i el node intentarà enviar diversos missatges al seu igual.

Les funcionalitats descrites es troben en els mètodes *Node_receive_message_version* i *Node.connect*.

7.2.8.5 Recepció del segon missatge *version*

Quan l'altre extrem rebi el segon missatge *version*, el node comprovarà si es tracta del d'una resposta i, en el cas afirmatiu, actualitzarà l'estat de la connexió a "DONE" i contestarà amb un messatge *VerAck*. En aquest punt, tal com ha succeït anteriorment, el node obtindrà la versió del protocol Bitcoin de l'altre parell. I, finalment, també es realitza el descobriment de veïns que veurem més endavant.

A partir d'aquest moment es considera que l'intercanvi de missatges (o *handshake*) s'ha finalitzat i els nodes es troben connectats.

Les funcionalitats descrites es troben en els mètodes *Node_receive_message_version* i *Node.connect*.

7.2.8.6 Recepció del missatge *VerAck*

Quan es rebi el missatge, el node comprovarà si té una connexió en estat "ESTABLISHED" amb l'origen del missatge. Si la connexió està establerta, s'actualitzarà a "DONE" i es marcarà al gestor d'adreces el node com a *connectat*. En cas contrari el node contestarà amb un missatge *reject*.

Les funcionalitats descrites es troben en els mètodes *Node._receive_message_verack*.

7.2.8.7 Tancament de la connexió

En cas que rebem un missatge *reject* o *close connection* (missatge que emularia el tancament de connexió TCP) d'una parell amb el que estem intentant establir una connexió (estat de la connexió *VERSION*), eliminarem l'intent de connexió del node. En la resta de casos el missatge s'ignora.

Les funcionalitats descrites es troben en el mètode *Node._receive_message_reject*.

7.2.9 TK-13: establir un mecanisme d'adreces concret

Problema
Pel funcionament del simulador, cal que els nodes es puguin comunicar entre sí mitjançant missatges adreçats a un identificador concret. Actualment els missatges per adreçar-se a un client empren el seu identificador.

D'altra banda, per tal d'implementar les funcionalitats per la gestió d'adreces de Bitcoin Core, cal que els nodes també es puguin identificar mitjançant un sistema similar a l'IP.

Anàlisi
Pel simulador cal fixar un mecanisme que determini com s'identifiquen els iguals de la xarxa del simulador. En aquest sentit es consideren dues alternatives:

- L'identificador del node, el sistema que funciona actualment.

- Una pseudo adreça IP, sistema necessari per implementar un comportament estocàstic similar al de l'agenda de Bitcoin Core.

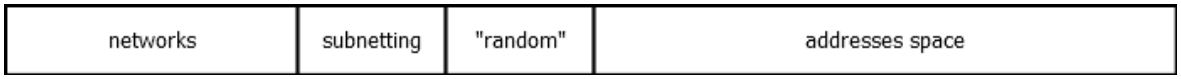
Per tal de decidir quina de les dues opcions implementar, s'han de considerar els següents punts:

- Els modes del simulador "xarxa estàtica" i "xarxa dinàmica definida" no necessiten una adreça IP, ja que no s'utilitzarà el gestor d'adreces (recordem que les connexions les estableix el simulador, no els nodes).
- Pel mode "xarxa dinàmica" sí que caldria algun tipus d'adreça IP, per aconseguir una entropia similar a la del gestor de direccions del client de referència. O, alternativament, es podria implementar un mecanisme estocàstic similar al de l'agenda de Bitcoin Core emprant, però, números enters en lloc d'adreces IPs.
- Actualment no s'assignen IPs en les simulacions.

Solució

En la implementació s'ha decidit utilitzar un sistema similar a IP per obtenir una major similitud al funcionament en el món real. Aquesta decisió comporta la necessitat d'establir un mètode que generi les adreces amb tres propietats: les adreces seran úniques, permetre que l'agenda d'adreces tingui un comportament estocàstic i, per últim, permetre que el sistema sigui escalable per suportar diverses xarxes.

Per garantir les tres condicions es va decidir implantar un sistema que permetés transformar de forma inequívoca l'identificador dels nodes a una adreça amb la següent estructura de les adreces:



Imatge 25: estructura d'una pseudo adreça IP en el simulador.

- *Networks*: espai d'un byte reservat per a les diferents xarxes que hi pugui haver en el simulador. A dia d'avui únicament existeixen tres xarxes que es poden comunicar entre si:
  - 1.0.0.0: reservada a nodes "DNS".
  - 2.0.0.0: reservada a nodes "Crawler".
  - 3.0.0.0: pels altres nodes.

  Es van separar aquestes tres xarxes per tal de poder diferenciar visualment de forma senzilla quin tipus de node ha enviat un missatge.
- *Subnetting*: espai de mig byte reservat per xarxes virtuals. Aquest espai no s'utilitza.
- *Random*: espai de mig byte reservat per un número "aleatori" que es calcula com al mòdul de 15 de la identitat del node.
- *Address space*: espai de dos bytes reservat per les adreces dels nodes.

I per calcular una adreça es realitza el següent procediment:

I. Es transforma l'identificador del node en una adreça IP mitjançant la següent operació:

```
ip = [str(identity >> (i << 3) & 0xFF) for i in range(0, 4)[::-1]]
```

Codi 29: generació d'una adreça IP en el simulador.

II. Es determina el tipus de node i se li assigna una xarxa modificant la IP.

III. S'afegeix aleatorietat en l'adreça del node mitjançant el càlcul del mòdul 15 de l'identificador del node. Es va prendre aquesta decisió per dos motius: el valor màxim de mig byte és de 16 (15 més 0) i, en segon lloc, és necessària aquesta aleatorietat per l'agrupament d'adreces que realitza l'agenda tal com es pot veure en el punt 5.5.2.

El codi implementat per dur a terme aquestes funcionalitats es troba al mètode *NodeProfiles*. *id_to_ip*.

7.2.10 TK-14: mecanisme per mantenir les connexions.

Problema
El client de referència de Bitcoin, per conèixer quines connexions són actives, disposa d'un mecanisme per determinar-ne la seva caducitat: és a dir, quan de temps fa que no es tenen notícies d'un node amb el que s'ha establert connexió i, quan el temps superi un llindar, la connexió "caducarà" i caldrà tancar-la.

El simulador no implementa aquesta característica.

Anàlisi
En el mode "descobriment de veïns", caldrà implementar les següents característiques:

A. Diferenciar, per cada connexió establerta, el seu estat en el mode "descobriment de veïns". Aquest sistema haurà de ser compatible amb els altres modes de funcionament del simulador. També haurà d'emmagatzemar quan s'ha realitzat l'última comunicació amb els iguals als que s'està connectats.
B. Un sistema periòdic, que es pot implementar en el mètode *periòdic_network*, que comprovi la caducitat de les diferents connexions establertes i procedeixi a tancar-les.
C. Cada cop que un node rebi un missatge d'un igual caldrà actualitzar la data de caducitat indicada en el primer punt.
D. S'haurà d'implementar un mecanisme que permeti enviar periòdicament els missatges *ping* i *pong* a tots els parells amb els que s'ha establert una connexió.

Solució
El codi implementat per resoldre aquest problema s'exposa en 7.2.8.2.

7.2.11 TK-15: mecanisme per inicialitzar la xarxa dels nodes.

Problema
En el capítol 5.3.1 exposàvem com Bitcoin Core realitzava el descobriment inicial de veïns (bootstrap) mitjançant dos sistemes: l'obtenció d'una llista d'adreces mitjançant servidors DNS o adreces d'altres nodes codificades en el propi programa.

El simulador no implementa completament aquestes funcionalitats.

Anàlisi
La inicialització de la xarxa dels nodes pot succeir en dos moments: en la primera execució del node i quan aquest es quedi sense adreces. En conseqüència es proposa dur a terme les següents millores:

A. Implementar un mecanisme per enviar peticions als nodes DNS quan sigui necessari. Per aquest punt serà necessari que els servidors DNS estiguin inicialitzats tal com s'ha exposat en l'apartat 7.2.4.
B. Controlar quan arribin aquests missatges i, si no arriben en un temps determinat emprar, els nodes codificats en el programa.

C. Per utilitzar els nodes codificats en el programa, caldrà implementar un mecanisme per obtenir-los.
D. Analitzar periòdicament si el node té adreces per tal de dur a terme la inicialització de la xarxa de nou si fos necessari.

Solució
Per la inicialització de la xarxa dels nodes s'ha implementat la inicialització dels servidors DNS (apartat 7.2.4) i de les adreces codificades en els nodes, un sistema que permeti utilitzar aquests dos serveis i la comunicació necessària entre els clients i els servidors DNS.

7.2.11.1 Inicialització de les adreces codificades en els nodes

La inicialització de les adreces codificades en els nodes es realitza de forma similar a la que realitzen els DNS exposada en l'apartat 7.2.4. En particular existeix la constant *Network. PERCENT_NODES_HARDCODED* que permet determinar, mitjançant un percentatge, el número màxim d'adreces codificades.

El mètode d'obtenció de les adreces és similar a l'exposat en "Codi 18: generació d'adreces per cada DNS (*Network. create_network_discovery*)", pel que no es reproduirà de nou.

D'igual forma, les adreces codificades en tots els nodes són les mateixes i es troben en la variable estàtica *Node.node_seeds*.

7.2.11.2 Ús de les adreces codificades en els nodes

Com que la inicialització de les adreces pot ser necessària en qualsevol moment, s'ha implementat la seva execució en el procés periòdic *Node.periodic_network.* Aquest procés, que s'executa amb una freqüència d'entre 0.0001 i 3[53], en primer lloc comprova si l'agenda d'adreces del node està buida i, si ho està, envia peticions als servidors DNS codificats en el programa (si n'hi ha algun).

Quan es rep resposta dels servidors DNS, o després d'un temps prudencial establert en la constant *Node.DNS_TIMEOUT*, es comprova si l'agenda disposa d'adreces gràcies als servidors DNS. Si no en disposa, s'incorporen les adreces codificades en el programa (si n'hi ha), que passen a formar part de l'agenda d'adreces dels nodes.

El control de la resposta dels servidors DNS s'implementa mitjançant dos semàfors emmagatzemats en *self.semaphores['dns_seeds']* (bloqueja l'execució del codi fins que hi hagi com a mínim una resposta vàlida o tots els DNS han contestat indicant que no disposen d'adreces) i *self.semaphores['dns_seeds']* (que controla el número de peticions que s'han enviat).

7.2.11.3 TK-15, TK-19: Transmissió de missatges de DNS

Per comunicar-se amb els servidors DNS, els nodes empren el sistema de missatgeria del simulador. En aquest sentit, com que els DNS no formen part del protocol Bitcoin, s'ha desenvolupat un mecanisme que permeti la comunicació directa amb els servidors (sense necessitat d'establir una connexió de Bitcoin mitjançant l'intercanvi de missatges *version*).

Concretament, quan es vol realitzar una connexió amb un servidor DNS es genera una nova classe *Link* (que és la que representa una connexió) amb l'estat *DNS*, que s'elimina tot seguit un cop el missatge s'ha enviat.

[53] L'aleatorietat s'ha implementat evitar que els nodes executin el manteniment de les connexions en el mateix moment.

De l'altra banda, quan el servidor DNS rep el missatge realitza el mateix procés: crea una nova connexió (objecte *Link*), envia un dels dos missatges de resposta possibles (*MessageDNSResponse*, si el servidor té adreces d'altres nodes, o *MessageDNSNone* en cas contrari) i finalment elimina la connexió creada després d'enviar el missatge.

Per últim, movent-nos de nou a l'altre extrem, quan el client rebi un missatge de resposta amb adreces, les afegirà a l'agenda i activarà el semàfor *self.semaphores['dns_seeds']* per tal que el programa continuï la seva execució. Aquest funcionament implica que el node pot seguir rebent respostes de DNS després d'haver finalitzat la inicialització de la xarxa –que es processaran normalment, afegint-les adreces a l'agenda.

En cas que el client rebi un missatge de resposta sense adreces, reduirà el valor de *self.semaphores['dns_seeds']* en u, i quan el valor d'aquest semàfor arribi a zero, es desbloquejarà l'altre semàfor (*self.semaphores['dns_seeds']*) per tal que el programa pugui continuar la seva execució.

```
elif isinstance(msg, ds.MessageDnsResponse):
    # Answer from a DNSQuery message
    # We only use the DNS in the dynamic network mode
    for address in msg.addr_list:
        self.addrmgr.update(address, msg.origin, self.env.now)
    self.log.debug([…])

    if len(msg.addr_list) > 0:
        # We've received seeds from a DNS, bootstrap complete.
        if 'dns_seeds' in self.semaphores:
            self.semaphores['dns_seeds'].succeed()

elif isinstance(msg, ds.MessageDnsNone):
    # Process the DNS semaphores
    if 'dns_seeds_num' in self.semaphores:
        self.semaphores['dns_seeds_num'] -= 1

        if self.semaphores['dns_seeds_num'] < 1:
            self.semaphores['dns_seeds'].fail(Exception("no answers"))
```

Codi 30: processament dels missatges DNS (*node.py, _process_received_message*).

7.2.12 TK-16: Implementar l'ús de l'agenda per part dels nodes.

Problema
El simulador únicament empra l'agenda d'adreces per obtenir adreces per realitzar una nova connexió (mètode *select*) i per obtenir adreces per un missatge *addr* (funció *get_random*). D'acord al que s'ha exposat en el capítol 5è, caldrà indicar a l'agenda quan s'intenta realitzar una connexió i quan s'ha aconseguit.

Addicionalment, la funció *get_random* torna una única adreça, el que obliga al node a invocar-la *n* cops per respondre un missatge *addr*.

Anàlisi
Per tal que els nodes emprin completament l'agenda d'adreces, caldrà implementar el següent:

A. Quan s'intenti una nova connexió a un igual, s'haurà de marcar a l'agenda.
B. Quan es realitzi una connexió s'haurà de marcar en l'agenda.

C. Cal implementar en l'agenda una funció per permeti obtenir *n* adreces de la mateixa, eliminant (o reaprofitant) el codi equivalent que realitza aquesta funcionalitat en la classe *Node*.

Solució
Pel que fa a la selecció de diverses adreces per respondre un missatge *getaddr*, aquesta funcionalitat estava implementada en la classe *Node*, en lloc de formar part de l'agenda d'adreces. En aquest sentit, es va moure i adaptar el codi a una nova funció denominada *NetAddressManager. get_address*.

S'han afegit una invocació a la funció *attempted* de l'agenda d'adreces en el moment que un node intenta establir una nova connexió en la funció *periòdic_network*. A més a més, també s'han afegit dues invocacions a la funció *connected* de l'agenda quan un node rep el segon missatge *version* o el missatge *VerAck*.

Les funcionalitats descrites es troben en la classe *Node*.

7.2.13 TK-17: mecanisme de descobriment de veïns.

Problema
Els nodes del simulador no implementen els mecanismes pel descobriment de veïns descrit en l'apartat 5.3.2.

Anàlisi
Al igual que en Bitcoin Core, els nodes del simulador hauran d'executar periòdicament els mecanismes necessaris per descobrir els altres iguals de la xarxa. En aquest sentit serà necessari implementar les següents característiques:

I. Quan un node estableixi una connexió d'entrada s'haurà d'anunciar, mitjançant la tramesa d'un missatge *addr*, i demanar adreces, a través d'un missatge *getaddr*, a l'igual amb el que s'ha connectat.

II. Amb caràcter periòdic els nodes s'hauran d'anunciar a tots els seus iguals. Per implementar aquesta funcionalitat caldrà que cada node controli de forma periòdica quan és l'últim cop que s'ha anunciat. Aquesta auto-promoció s'implementarà mitjançant la tramesa d'un missatge *addr* que únicament contindrà l'adreça del node.

   Cal destacar que, tal com s'ha exposat en el punt 6.2.1.3, el sistema de missatgeria del simulador divergeix de Bitcoin Core. En el client de referència, cada cop que es vol enviar una adreça a un igual, s'emmagatzema en un buffer i, periòdicament (o davant d'un missatge *getaddr*) es trameten.

III. Cada cop que un node rebi un missatge *addr* caldrà que reenviï les adreces que rep a dos dels iguals amb els que ha establert una connexió. Aquests dos nodes seran els mateixos durant un període de temps determinat.

Solució
En l'apartat 7.2.13 s'han definit tres mecanismes que el simulador ha d'implementar per dur a terme el descobriment de veïns: anunciar-se i sol·licitar adreces en el moment de realitzar la connexió, anunciar-se periòdicament a totes les connexions i reenviar totes les adreces als mateixos dos nodes durant un període de temps prefixat.

Per dur a terme la primera funcionalitat, s'ha implementat el mètode *Node_receive_message_version_announcement*, invocat poc després de realitzar la

connexió, que tramet un missatge *addr* que conté únicament l'adreça del node que l'envia i un missatge *GetAddr* demanant adreces.

Pel que respecta a l'anunci periòdic del node a totes les adreces, en la funció *Node.periodic_network*, s'ha implementat el següent codi:

```
# Self announcement in the network
now = self.env.now
if (last_rebroadcast + Node.SELF_ANNOUNCEMENT) > now:
    msg = ds.MessageAddr(origin=self.get_address,
                         destination=None,
                         addr_dict={self.get_address: now},
                         timestamp=now)
    self.log.debug("%7.4f: Node |%d| has announced itself to their "
                   "connections with a broadcast message. " %
                   (now, self.id))
    last_rebroadcast = now
# End self announcement
```

Codi 31: anuncis periòdics en la xarxa

Tal com es pot veure el node emmagatzema en la variable *last_rebroadcast* l'últim moment en que s'ha realitzat la auto-promoció i en *Node.SELF_ANNOUNCMENT* la freqüència en que es realitza. El mètode concret és el mateix que s'empra en el punt anterior, quan s'envia la pròpia adreça en un missatge *addr*.

Per últim, cada cop que es rep un missatge *addr* es reenvien totes les seves les adreces que contingui i que siguin vàlides. Per aconseguir-ho s'ha d'aconseguir dos punts: seleccionar dos nodes, entre els que estiguem connectats, durant un període de temps determinat i reenviar les adreces a aquests dos nodes cada cop que es rep un missatge *addr.*

D'altra banda, per implementar la selecció dels nodes, s'ha desenvolupat en la funció *Node.maintain_link* una iteració entre tots els nodes amb els que s'ha establert una connexió. En aquest mètode es comprova que hi hagi seleccionats els dos nodes als que reenvien les adreces (i si cal canviar els nodes ho realitza).

Es pot discriminar aquelles connexions a les que es reenvien les adreces perquè la variable *Link.forward_addresses* no és null i hi figura quan s'ha seleccionat l'adreça per efectuar el reenviament.

7.2.14 TK-18: Els nodes DNS funcionen sobre el protocol Bitcoin

Problema
En la implementació actual del simulador tots els nodes, inclosos els servidors DNS, necessiten establir una connexió de Bitcoin per intercanviar missatges, amb les limitacions i característiques abans mencionades. Actualment els nodes DNS intenten establir una connexió de Bitcoin abans d'enviar missatges DNS.

Aquest comportament no és necessari, ni succeeix en el món real, ja que els servidors DNS no formen part de Bitcoin.

Anàlisi
Tal com s'ha exposat abans, com que els servidors no formen part de la xarxa Bitcoin, caldria permetre la connexió en qualsevol cas. Per aquest motiu es considera que el simulador hauria d'implementar les següents funcionalitats:

A. Per la comunicació entre nodes i servidors DNS no serà necessari establir una comunicació de Bitcoin.
B. Conseqüentment, les connexions establertes entre un node i un servidor DNS no es considerarà pel que fa al número màxim de connexions que pot assumir el node.

Solució

La solució s'ha descrit en l'apartat 7.2.11.

7.3 Conclusions

Al llarg d'aquest apartat hem pogut veure les diferents solucions que s'han implementat al BTC-NS per tal que l'eina de simulació tingui un comportament més aproximat al client de referència Bitcoin Core. En base a les modificacions realitzades en la secció **¡Error! No se encuentra el origen de la referencia.** proposem un conjunt de millores que es poden dur a terme. Així mateix, al final d'aquest apartat exposarem algunes de les característiques rellevants del simulador.

Pel que fa a la depuració i tests dels canvis implementats, tal com s'ha descrit s'ha utilitzat el marc de treball *unitest* per realitzar les proves unitàries i, per aquelles aspectes en que no era senzill aplicar-lo (degut a SimPy), s'ha emprat el mode de depuració de l'entorn de programació per analitzar el comportament del programa.

Si recapitulem els canvis implementats els podem resumir en la següent taula:

Id.	Descripció	Solució implementada
TK-01	Modificar el nom de la classe *BitnodesCrawler* a *Crawler*.	Canviades les referències.
TK-02	Eliminar la variable *port* del gestor d'agendes.	Eliminada la variable.
TK-03	Eliminar les variables *addr_know* i *addr_to_send* de l'agenda.	Eliminades les variables i funcions relacionades.
TK-04	Implementar la latència de xarxa en el sistema de missatgeria el simulador.	Descrit en 7.2.1.
TK-05	Emmagatzemar la informació necessària de l'aranya.	Descrit en 7.2.2.
TK-06	Dissenyar i implementar un sistema de missatgeria únic.	Descrit en 7.2.3.
TK-07	Establir un mecanisme per transmetre un missatge entre tots els nodes amb els que s'ha establert comunicació.	En consonància amb l'anterior, s'emprarà el mètode *send_broadcast_message*.
TK-08	Analitzar les característiques estocàstiques de l'agenda del simulador.	Descrit en 7.2.4.
TK-09	Implementar un mecanisme per inicialitzar les dades del node DNS.	Descrit en 7.2.5.
TK-10	Implementar un mecanisme per crear nodes en el mode "descobriment de veïns".	Descrit en 7.2.6.
TK-11	Establir un mecanisme per tal que l'aranya analitzi periòdicament la xarxa.	Descrit en 7.2.7

Id.	Descripció	Solució implementada
TK-12	Dissenyar el mecanisme d'establiment de connexió entre els nodes en el mètode de "descobriment de veïns".	Descrit en 7.2.8.
TK-13	Establir un mecanisme d'adreces concret..	Descrit en 7.2.9.
TK-14	Implementar un mecanisme per tal que els nodes puguin mantenir les connexions.	Descrit en 7.2.10.
TK-15	Establir un mecanisme per tal que els nodes puguin inicialitzar la xarxa.	Descrit en 7.2.11.
TK-16	Implementar l'ús de l'agenda per part dels nodes.	Descrit en 7.2.12.
TK-17	Implementar el mecanisme de descobriment de veïns.	Descrit en 7.2.13.
TK-18	Els nodes DNS no funcionen de forma externa al mecanisme de connexions de Bitcoin.	Descrit en 7.2.14.

Taula 23: resum de les millores implementades en el simulador

Amb els canvis abans descrits, més alguna correcció d'errors que no s'ha esmentat, es poden donar per implementades les millores necessàries per tal que el simulador tingui un funcionament similar al client de referència de Bitcoin i, per tant, procedir a l'estudi de l'eficàcia d'una aranya que veurem en el proper capítol.

No obstant a l'anterior, cal considerar que hi ha una gran diferència entre Bitcoin Core i el simulador: el primer és un sistema que funciona en temps real, mentre que el segon treballa en un temps finit emprant "unitats de temps" simulades. En aquest sentit, tot i que creiem que el comportament serà similar, s'haurà d'analitzar en el proper capítol aquest fet.

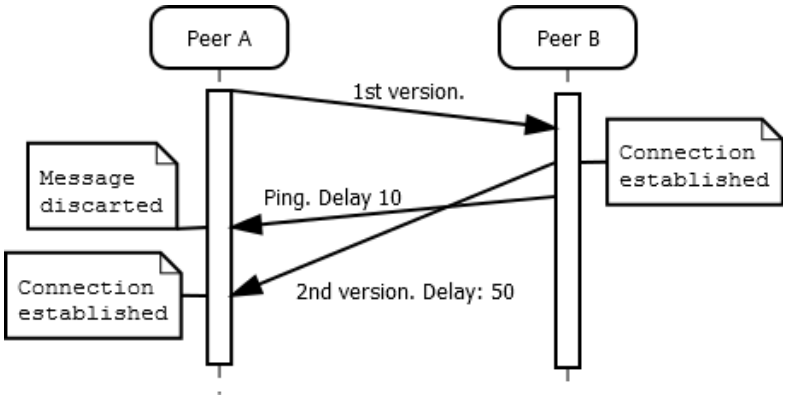
7.3.1 Altres característiques del simulador

Per tal de dur a terme el proper capítol, caldrà considerar diversos aspectes del comportament del simulador que, tot i que no estan relacionats amb el comportament del client de referència, són necessaris pel entendre el seu funcionament, ja que tindran un impacte en les simulacions.

7.3.1.1 Transmissió de missatges

Tal com hem exposat anteriorment, per tal de transmetre un missatge, cal que la connexió Bitcoin estigui establerta. En aquest sentit, degut a que un missatge tarda un temps a viatjar entre dos iguals, en ocasions un node pot rebre un missatge abans que s'estableixi la connexió.

Per exemple, i tal com podem veure en la següent imatge, l'igual *B* considera que ha establert la connexió amb el node *A* en el moment que ha enviat el segon missatge *version* i, per tant, un cop enviat el missatge li tramet correctament una comunicació *ping*. No obstant, el node *A* únicament considerarà establerta la connexió quan rebi el missatge *version*. En conseqüència si, tal com succeeix en la il·lustració, *A* rep el missatge *ping* de *B* abans de que li arribi el missatge *version,* el descartarà al considerar que la connexió no s'ha establert.



Imatge 26: afectació del retard a l'acceptar un missatge en el BTC-NS.

8 Estudi de l'eficàcia de l'aranya

En el present capítol exposarem l'execució de l'últim objectiu del projecte: l'estudi de l'eficàcia d'una aranya per determinar la topologia de la xarxa. Tal com s'ha mencionat anteriorment, l'anàlisi del comportament de l'aranya es realitzarà mitjançant el BTC-NS, una eina de simulació que permet emular el comportament d'una xarxa d'iguals Bitcoin.

Per tal de dur a terme els treballs, en primer lloc, cal considerar que el simulador no s'ha provat en estudis d'aquestes característiques, pel que els valors de configuració inicials (exposats en el proper apartat) són subjectius, i, per tant, és possible que les simulacions realitzades amb la configuració per defecte produeixin resultats incorrectes.

En aquest sentit, s'ha de considerar que, tot i que s'ha traslladat al simulador el funcionament del client de referència de Bitcoin, determinades constants que regulen el seu comportament no es poden importar directament, ja que el seu marc temporal és diferent (Bitcoin Core treballa en temps real, mentre que el simulador ho fa en temps discret), i caldrà ajustar-les estudiant com funciona la xarxa del simulador.

En conclusió, per analitzar l'eficàcia de l'algorisme de l'aranya implementat en el BTC-NS, caldrà executar diverses simulacions, establint diversos valors de configuració, per tal d'assegurar que els resultats no estiguin condicionats en excés pel comportament del simulador i, per tant, poder assegurar de certa manera l'exactitud de l'estudi.

Així mateix, cal remarcar que el desenvolupament de diversos algorismes d'aranya queda fora de l'àmbit del projecte, pel que en l'estudi que durem a terme ens limitarem a analitzar l'aranya amb el comportament definit al simulador sense realitzar-hi modificacions o millores.

Per últim, en un altre ordre de coses, per tal de dur a terme l'estudi i que aquest sigui reproduïble, al llarg del capítol establirem la configuració del simulador que regularà el seu comportament (apartat 8.1), determinarem la metodologia de l'estudi (apartat 8.2), exposarem els resultats i incidències de l'execució (apartat 8.3) i, per últim, detallarem les conclusions finals (capítol 10).

8.1 Configuració de les simulacions

En primer lloc procedirem a identificar la configuració sobre la que s'executaran les simulacions. En concret definirem la versió emprada del simulador, les opcions de configuració inicials, ja siguin definides per l'usuari o incloses en la programació del simulador, i, per últim, el mètode d'execució de les simulacions.

Pel que fa a la versió del simulador, atenent que encara no disposa d'una política de oficial de versions, emprarem el número de revisió del repositori Subversion que, en concret, és la revisió 335.

Respecte als valors de configuració establerts per l'usuari són els següents:

Atribut	Descripció	Valor
Mode del simulador	"normal" o "avançat"	"normal"
Temps d'execució	Unitats de temps (en el simulador) en el que s'executarà.	10.000ut[54]

[54] "ut" per "unitat de temps" en el simulador.

Atribut	Descripció	Valor
Número de nodes	-	1.000
Tipus de nodes	Perfil i percentatge de nodes que participaran en la simulació	DNS: 01% Node: 99% Aranya: 01 unitat
Mode de connexió	Entre “xarxa estàtica”, “xarxa dinàmica predefinida” o “xarxa dinàmica”.	“xarxa dinàmica”
Atributs a simular	Els atributs que s’ignoraran en la simulació.	block_dummy tnx_details
Dades a emmagatzemar en BD	Les dades que es guardaran en base de dades	EVENT_MSG_SENT EVENT_MSG_RECEIVED EVENT_CONNECTION_CREATED EVENT_NODE_ADDED
Ample de banda	El model per obtenir l’ample de banda dels nodes.	Mode: normal Mitjana: 2 Desviació: 2
Latència de xarxa	El model per obtenir la latència entre les connexions dels iguals.	Mode: constant Valor: 3ut

Taula 24: configuració d’usuari per les simulacions

I, d’altre banda, els valors establerts en les constants programades en el simulador són:

Classe	Constant	Ús	Valor
Link	NONE_TIMEOUT	Temps que es tarda a eliminar una connexió no establerta	300ut
	VERSION_TIMEOUT	Caducitat en l’establiment d’una connexió	600ut
	ESTABLISHED_TIMEOUT	Caducitat en una connexió establerta	600ut
	FORWARD_ADDRESSES_MAX	Número màxim d’adreces a les que es reenviaran els missatges *addr* rebuts.	2 nodes
	FORWARD_ADDRESSES_TIME	Temps durant el qual totes les adreces es reenviaran a un node en concret	200ut
	FORWARD_ADDRESSES_YOUNGER	Únicament es reenviaran les adreces més “joves” que la constant	200ut
Network	PERCENT_NODES_DNS	Percentatge màxim d’adreces que serviran els nodes DNS	25%
	PERCENT_NODES_HARDCODED	Percentatge màxim d’adreces que tindran codificades tots els clients	10%
NetAddress Manager	ADDRMAN_BUCKET_SIZE	Mida dels sacs en el gestor d’adreces	64
	ADDRMAN_TRIED_BUCKET_COUNT	Número de sacs “provats”	256
	ADDRMAN_NEW_BUCKET_COUNT	Número de sacs “nous”.	1024
	ADDRMAN_TRIED_BUCKETS_PER_GROUP	Número de sacs per agrupació d’adreces del rang /16.	8
	ADDRMAN_NEW_BUCKETS_PER_SOURCE	Número de sacs per agrupació d’adreces d’acord a la font.	64
	ADDRMAN_GETADDR_MAX_PCT	Percentatge d’adreces màximes que es respondrà en un missatge *addr.*	23%

Classe	Constant	Ús	Valor
NetAddress Manager	ADDRMAN_GETADDR_ MAX	Número màxim d'adreces que es contestarà en un missatge *addr*.	2500
Node	DNS_TIMEOUT	Caducitat en la resposta a un node DNS	350ut
	SELF_ANOUNCEMENT	Període en el que el node s'anuncia als iguals de la xarxa amb els que està connectat.	350ut
	PING_ANOUNCEMENT	Període en el que el node envia missatges *Ping*.	175ut
	MAX_OUTBOUND_ CONNECTIONS	Número màxim de connexions de sortida concurrents	2
	MAX_CONNECTIONS	Número màxim de connexions.	8
	Connexions[55]	El node manté les seves connexions durant tota la simulació	Sí

Taula 25: configuració programàtica del simulador

Per últim, pel que fa als ordinadors que executaran les simulacions, s'han emprat dues instàncies al núvol: una amb 2 nuclis i 32 GB de RAM i la segona amb 4 nuclis i 15 GB de RAM. Ambdues màquines virtuals funcionen amb una distribució Debian GNU/Linux 8. Per executar la simulació es modificaran els valors oportuns de configuració del fitxer *simulation.py*, d'acord a les taules anteriors, i s'invocarà amb la següent comanda (on "[...]" correspon a la ruta del fitxer):

```
$ nohup /usr/bin/python2.7 […]/simulation.py &
```

Codi 32: comanda d'execució del simulador

Nohup, és una comanda que permet dissociar una instrucció concreta de la consola des d'on s'executa, permetent que es tanqui sense acabar l'execució del programa. I, pel que respecta a l'opció "&", indicada al final de la comanda, ordena que la instrucció s'ha d'executar en segon pla.

La configuració de les instàncies al núvol és relativament senzilla, ja que únicament necessita un interpretador de la versió 2.7 de Python, un servidor Mysql i instal·lar els requeriments del projecte mitjançant l'eina *pip*, que permet instal·lar tots els components de Python necessaris[56], amb la següent instrucció (on *requeriments.txt* és un fitxer que es troba en el codi font del simulador):

```
# pip install –r requeriments.txt
```

Codi 33: comanda per instal·lar dependències Python

D'altre banda, per configurar Mysql caldrà crear una base de dades i un usuari amb les següents instruccions (en el simulador les dades de connexió es troben en el fitxer *simulation.py*):

[55] No es tracta d'una constant, es va desactivar el codi que desactivava aleatòriament les connexions.

[56] També caldrà instal·lar programari divers, com el controlador de Mysql per Python, mitjançant el gestor de paquets del sistema operatiu. El propi *pip* indica els paquets a instal·lar.

```
mysql> CREATE DATABASE mydb;
mysql> CREATE USER 'bt-ns'@'localhost' IDENTIFIED BY 'bt-ns';
mysql> GRANT ALL on mydb.* TO 'bt-ns'@'localhost';
```

Codi 34: comandes per generar l'usuari i la BD en Mysql

Finalment, caldrà importar l'estructura de taules mitjançant la següent comanda (on *create_tables.sql* és un fitxer que es pot trobar a l'estructura de directoris del simulador):

```
$ mysql –u root –p < create_tables.sql
```

Codi 35: comandes per importar l'estructura de taules

8.2 Metodologia de les simulacions

L'eficàcia d'una aranya en determinar la topologia d'una xarxa de parells està influïda per dos factors: d'una banda la capacitat del simulador per emular la xarxa de parells de Bitcoin i, d'altre banda, el comportament de la pròpia aranya en el sistema (és a dir, quin algorisme utilitza per descobrir iguals).

En el context del present projecte, tal com hem exposat en la introducció, ens limitarem a l'estudi de l'aranya d'acord a l'algorisme implementat al simulador sense introduir-hi millores o canvis; el que ens suposarà una avantatge: permetrà avaluar si el comportament de l'aranya està influenciat pel simulador i, en cas afirmatiu, fins a quin punt.

El comportament de l'aranya, serà el descrit i implementat en la secció 7.2.7 que, en resum, és el següent:

- Quan l'aranya obté una adreça la classifica en quatre blocs: adreces "noves" (amb les que no ens hem connectat), adreces en procés de connexió, adreces "provades" (amb les que ens hem connectat) i adreces "dolentes" (aquelles amb les que no s'ha pogut establir una connexió). Per seleccionar una direcció a la que connectar-se, es procedeix a escollir-la aleatòriament del primer grup que en contingui alguna[57]. Cal concretar que la selecció és completament aleatòria i només es comprova el temps des de l'última connexió (l'aranya evita connectar-se diversos cops al mateix node en un interval curt).

- Així mateix, cada cop que es realitza una connexió, l'aranya envia un missatge *GetAddr* per tal d'obtenir més adreces.

- D'altre banda, únicament ens centrarem en les adreces recollides per l'aranya, sense entrar en consideració en si l'aranya es pot connectar amb l'igual.

- Per últim, l'aranya tancarà una connexió quan hagi rebut un missatge *Addr* (que serà la resposta al *GetAddr* anterior) o bé quan caduqui la connexió.

Pel que fa a l'estudi en si, l'enfocarem en la capacitat de l'aranya per conèixer la topologia de la xarxa, centrant-nos en el número d'adreces obtingudes, sense considerar altres elements com si els clients estan actius o el tipus o versió del client. Per calcular l'eficiència emprarem el següent fórmula:

[57] Excepte de les adreces amb les que s'està establint una connexió.

$$Eficàcia\ de\ l'aranya = \frac{Número\ d'adreces\ descobertes}{Número\ total\ de\ nodes}$$

D'altre banda, tal com s'ha exposat, durem a terme diverses simulacions on, per cada una d'elles, ajustarem els valors de configuració d'acord als resultats de la simulació anterior i, pel que respecta a la primera simulació, emprarem les opcions d'execució contemplades en l'apartat 8.1.

Respecte a l'obtenció i presentació de les dades de la simulació, les exposarem en un conjunt de taules, on mostrarem la informació extreta agrupada en nodes, missatges, l'aranya i les connexions realitzades. I, per cada una d'aquestes taules, procedirem a estudiar els seus valors, per identificar millores i extreure'n conclusions.

Les dades que emprarem s'obtindran mitjançant l'explotació semi-automàtica dels esdeveniments generats en la simulació, que s'emmagatzemen en una base de dades relacional MySql, i dels fitxers de depuració proporcionats pel propi simulador (que permeten disposar d'informació "en format humà" sobre l'execució).

Per últim, per assegurar la qualitat dels resultats, cada simulació s'executarà en dues màquines amb perfils diferents (una té més RAM però menys processador i l'altre el contrari) però amb el mateix sistema operatiu i la mateixa configuració. Amb aquesta doble execució es pretén assegurar la qualitat dels resultats i que no estan limitats per les característiques de maquinari.

8.3 Execució i resultats

Per la realització de l'estudi realitzarem tres simulacions que exposarem tot seguit.

8.3.1 Primera simulació

En aquesta simulació s'ha configurat el programari amb els valors per defecte que apareixen en Taula 24 i Taula 25 (pàgina 89) i els seus resultats són els següents

Dades genèriques

Primer de tot analitzarem les "dades genèriques" de la simulació:

Valor	Resultats / Valors		Observacions
	Simulació I.A	Simulació I.B	
Característiques de maquinari	2 CPU[58] 30 GB RAM	4 CPU 15 MB RAM	
RAM màxima	10,5 GB	5.5 GB	Incloent la memòria consumida per la base de dades i el s.o. No s'executaven altres processos de rellevància.
RAM mínima	7,4 GB	4,06GB	
Ús CPU màxim	100%	100%	Ús de la CPU en el post processament.
Ús CPU mínim	43.2%	38.6%	Ús de CPU durant la simulació.
Temps d'execució	9,91 hores	9,08 hores	Execució de la simulació.

[58] Al tractar-se d'instàncies al núvol, són processadors virtuals i no podem proporcionar informació sobre la seva arquitectura, marca o velocitat.

Valor	Resultats / Valors		Observacions
	Simulació I.A	Simulació I.B	
Temps de post-processament[59]	>12 hores	> 12 hores	Es va avortar abans d'acabar.
Mida de la BD	471 MB	472 MB	Mida de la BD (en format de text, com a resultat d'exportar-la amb *mysqldump*)

Taula 26: dades genèriques de la 1a simulació

Com podem veure en la taula anterior, el temps de processament durant la simulació (deixant, de banda el post processament de les dades) no és directament proporcional a la capacitat de la CPU. En aquest sentit, podem observar com duplicant la potència de càlcul únicament hem disminuït el temps d'execució en menys d'un 10%.

Aquest fet es deu a dos motius: l'alt volum de missatges tramesos en el simulador (tal com veurem en el proper apartat) i l'ús de crides síncrones en la base de dades. Aquesta observació es pot corroborar si s'executa la mateixa simulació sense connexions a base de dades. En aquest cas l'ús de CPU és manté constant al 100% i s'executa en unes 7 hores.

D'altre banda, podem comprovar com, per dur a terme la simulació, es necessitarien, com a mínim, uns 4 GB de memòria RAM exclusius pel procés de simulació més un altre per la base de dades MySql. En conclusió, la màquina on s'executi una simulació d'aquestes característiques hauria de disposar com d'uns 5~6 GB de RAM.

Dades dels nodes

Pel que fa als valors relacionats amb els nodes són:

Valor	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Número de nodes	1.000		
Número *nodes convencionals*	899		
Número de DNS	100		
Número d'aranyes	1		
Bootstrap			
Ús de DNS?	Sí		S'han usat els servidors DNS per arrancar?
Número d'adreces per DNS	190		Número mig d'adreces per servidor DNS (valor aproximat)
Nodes que els han emprat	899		Número de nodes que han emprat l'arrancada per DNS
Ús d'adreces codificades?	Sí		S'han emprat les adreces codificades en el programa?

[59] Recordem que "el post-processament" és una tasca del simulador que es realitza després de la simulació i que analitza les dades. En aquest estudi no l'utilitzarem.

Valor	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Número d'adreces codificades	10		Número d'adreces codificades en els nodes
Nodes que les han emprat	892		Número de nodes que han emprat l'arrancada amb les adreces codificades en el programa

Taula 27: valors dels nodes en la 1a simulació.

D'aquesta taula podem veure com tots els nodes han realitzat l'arrancada de la xarxa emprant els servidors DNS (que és el comportament esperat) però, a banda d'això, prop del 99% dels nodes també han emprat els nodes codificats en el propi programa, el que és considera sorprenent ja que no hauria de succeir en un percentatge tant alt.

En aquest sentit, si procedim a analitzar el retard dels missatges DNS, podem observar com els valors en els missatges *DnsQuery* són "normals" (entre 20 i 200 unitats de temps), però són sorprenentment alts en els missatges *DnsResponse*, on es superen de llarg les 1.000 unitats de temps (tal com es mostra a tall d'exemple en la següent imatge).

```
|194| will send |186| addresses to |50791352|
(IP |17694914|) sent message |DnsResponse| to node |952| (IP |50791352|). Delay: |35469.27|.
|282| will send |201| addresses to |50791337|
(IP |17563930|) sent message |DnsResponse| to node |937| (IP |50791337|). Delay: |5732.23|.
|237| will send |203| addresses to |50463562|
(IP |17563885|) sent message |DnsResponse| to node |842| (IP |50463562|). Delay: |10488.85|.
|274| will send |193| addresses to |50397185|
(IP |17039634|) sent message |DnsResponse| to node |1| (IP |50397185|). Delay: |8313.71|.
|699| will send |202| addresses to |50725800|
(IP |17367739|) sent message |DnsResponse| to node |936| (IP |50725800|). Delay: |2877.37|.
|44| will send |208| addresses to |50463232|
IP |17694764|) sent message |DnsResponse| to node |512| (IP |50463232|). Delay: |1515.44|.
|884| will send |193| addresses to |50987468|
(IP |17695604|) sent message |DnsResponse| to node |460| (IP |50987468|). Delay: |22114.44|.
```

Imatge 27; cost temporal elevat en la tramesa de missatges DNS.

Aquest fet, que no s'havia apreciat en les simulacions de prova, es deu al mètode de càlcul de la mida dels missatges exposat en el Codi 16. Aquesta funció torna la mida d'un missatge computant quan ocupen les seves variables en memòria i, posteriorment, aquest valor es divideix per l'ample de banda del node per obtenir el retard en la transmissió d'un missatge.

Com que les comunicacions *DNSResponse* contenen prop de 190 adreces IP, la seva mida és desproporcionada en relació a l'ample de banda dels nodes, el que suposa que en alguns casos s'obtingui un retard superior al temps d'execució de la simulació.

Per donar solució a aquest problema s'ha modificat el mètode de càlcul de la mida d'un missatge tal com es mostra a continuació. Amb aquesta modificació, en la propera simulació, veurem com el problema desapareix però, d'altre banda, la reducció considerable en el temps de tramesa dels missatges implicarà que caldrà plantejar –en un futur– si es disminueixen les constants que regulen la caducitat de les connexions de la simulació.

```
def size(self):
    size = 0
    for attr in vars(self):
        if not attr.startswith("__"):
            if isinstance(attr, int):
                size += attr / 65535
            elif isinstance(attr, float):
                size += (int(attr) / 65535) + 1
            elif isinstance(attr, str):
                size += len(attr) / 2
            elif isinstance(attr, list):
                size += len(attr) * 2
            elif isinstance(attr, dict):
                size += len(attr) * 4
            elif isinstance(attr, decimal.Decimal):
                size += 2
    return size
```

Codi 36: nova implementació del càlcul de la mida del missatge.

En segon lloc també podem apreciar com d'una simulació de 1.000 nodes, el 10% són servidors DNS i cada un dels servidors conté prop de 190 nodes (el 25% aproximadament). Aquest comportament es deu, respectivament, a un error de configuració d'usuari (en lloc de l'1% estipulat en la Taula 24 es va introduir el 10%) i a la constant PERCENT_NODES_HARDCODED que està establerta al 25% dels nodes.

Dades dels missatges

Valor	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Total de missatges	1.441.680	1.458.195	Número de missatges enviats en el simulador. Únicament es contemplen els missatges relacionats amb la xarxa. La base de dades recull dos cops el missatge: quan s'envia i quan es rep, aquestes dades corresponen a un dels dos esdeveniments.
msg. *addr*.	1.171.934	1.186.695	
msg. *close*.	23.545	23.876	
msg. *DNS none*	0	0	
msg. *DNS query*	89.900		
msg. *DNS response*	89.719	89.742	
msg. *GetAddr*	3.200	3.193	
msg. *Reject*	40.820	41.770	
msg. *VerAck*	3.200	3.193	
msg. *Version*	19.362	19.484	

Taula 28: valors estadístics dels missatges enviats en la 1a simulació.

D'aquests valors podem observar com prop del 80% dels missatges són *addr*, el que representaria un valor molt alt, però que és comprensible pels següents factors:

- El valor de la constant SELF_ANNOUNCEMENT està establert en 350ut, pel que s'executarà 28 cops. Cada cop que s'executi s'enviarà un missatge *addr* a tots els nodes amb els que s'ha establert connexió.
- Cada cop que s'estableixi una connexió s'enviarà un altre missatge.

- S'enviaran cada cop que es rebi un missatge *GetAddr*.
- Cada cop que un node rebi un missatge *addr*, tal com s'ha exposat en l'apartat 7.2.13, el reenviarà a un o dos nodes mitjançant nous *addr*.

Davant d'aquest comportament es proposa realitzar les següents modificacions:

- S'hauria d'augmentar la constant SELF_ANNOUNCEMENT ja que no compensa la càrrega de treball d'una freqüència tant alta amb la poca informació que proporciona als nodes de la simulació. Així dons l'establiríem en 600ut o bé que fos un 10% del temps de simulació (pel que s'executaria 10 cops o menys).
- Es podria afegir als missatges un identificador per conèixer des de quin punt de la simulació s'han generat, el que permetria obtenir informació concreta sobre on es creen i es podrien dur a terme els anàlisis necessaris per tal d'ajustar el comportament del simulador. Aquest punt el contemplem en el capítol 9, millores futures.

En segon lloc podem veure com els missatges *close* ens indiquen que hi ha prop de 23.500 connexions que s'han tancat. Al igual que abans, es considera que seria d'interès establir el motiu del tancament de connexions.

També podem observar com no s'han enviat missatges del tipus *DNS None*, el que es considera normal ja que tots els servidors DNS estan programats perquè funcionin correctament i tinguin adreces amb les que respondre.

En quart lloc detectem que el número de missatges *DNS Query* no coincideix amb les respostes proporcionades pels servidors DNS (la suma dels missatges *DNS None* i *DNS Response*). Aquest fet es deu que, tal com apreciem en la Imatge 27, alguns dels missatges tenien un retard superior al temps de simulació.

Pel que respecta als missatges *GetAddr* coincideixen amb els missatges *VerAck* (els missatges *GetAddr* s'envien conjuntament amb *Verack* en tots els casos, incloent l'aranya).

En sisè lloc, pel que fa els missatges *Reject*, com que ens proporcionen el motiu pel que s'envien, podem obtenir informació addicional que pot ser d'interès. Per exemple en la simulació I.B podem veure com menys del 0,1% dels missatges es deu a que un node intenta establir una connexió que, per l'altre extrem, ja existeix; el 71% a que arriben missatges abans d'establir una connexió (la característica de funcionament identificada en la secció 7.3.1.1) i el 28% restant a que s'intenten connexions amb nodes que ja han arribat al seu límit.

reason	(COUNT(*)/2)
Already connected	23.0000
Connection not established	29843.5000
Couldn't accept more connections	11904.0000

Imatge 28: causes dels missatges *Reject* en la Simulació II

Com que la principal causa del rebuig de missatges és el fet que arriben abans d'haver-se iniciat la connexió (casuística que sospitem que no succeeix en el món real en aquesta proporció) seria convenient que, tots els missatges que s'enviïn quan el node hagi establert connexió, disposin de un retard que asseguri que no arribaran abans que el client, a l'altre banda de la connexió, sigui coneixedor que s'ha completat el protocol de *handshake*. Aquesta millora l'identifiquem en el capítol 9.

Per últim, però el fet més important de tots, és que en total s'han generat prop d'un milió i mig de missatges en la simulació. Això implica que, com que els missatges es guarden dos cops en base de dades, s'han realitzat prop de tres milions d'insercions individuals a la base de dades MySql. I, en conseqüència, hi ha un enorme cost temporal degut a la base de dades que es podria millorar de quatre formes diferents (que són compatibles entre sí):

I. Establir un sistema intermediari que agrupi les diverses insercions individuals en la base de dades i les trameti conjuntament tal com es proposa en l'apartat 9.4
II. Procurar evitar guardar dos cops el mateix missatge tal com es proposa en l'apartat 9.5.
III. Realitzar una administració de la base de dades MySql per tal d'intentar millorar-ne la velocitat.
IV. Reduir el número de missatges tramesos.

Dades de l'aranya
Pel que fa als missatges que ha tramés l'aranya durant la simulació (i estan inclosos en la Taula 28 de l'apartat anterior), els valors són:

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Missatges enviats			
Núm. enviats *close*.	9	11	
Núm. enviats *GetAddr*	8	6	
Núm. enviats *Reject*	8	22	
Núm. enviats *VerAck*	8	6	
Núm. enviats *Version*	9		
Núm. rebuts *Addr*.	16	26	
Mitjana d'adreces rebudes per *Addr*	1		
Número d'adreces rebudes diferents	9	15	Inclou l'adreça de l'aranya
Cops que l'aranya rep la seva adreça	8	4	
Missatges rebuts			
Núm. rebuts *close*.	7	6	
Núm. rebuts *GetAddr*	0		
Núm. rebuts *Reject*	2	4	
Núm. rebuts *VerAck*	0		
Núm. rebuts *Version*	14	13	

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Informació sobre els nodes			
Nodes coneguts per l'aranya	17	23	Incloent les adreces codificades i descartant les direccions duplicades i l'adreça de l'aranya.
Adreces codificades en l'aranya	10		

Taula 29: valors estadístics dels missatges enviats i rebuts per l'aranya en la simulació.

Podem qualificar els números anteriors com una decepció ja que no són, ni de bon tros, els esperats. Concretament, tot i que sembla que l'aranya ha funcionat durant tota l'execució de la simulació no ha rebut les adreces suficients de la xarxa per dur a terme la seva funció. Aquestes paraules es veuen reforçades pels següents fets:

A. El baix número de missatges *addr* rebuts.
B. La majoria de missatges *addr* contenien una única adreça –que en diverses ocasions era la pròpia direcció de l'aranya.
C. El baix número de connexions de sortida.
D. Tal com indiquem en l'apartat 0, el 99% dels nodes han alimentat la seva agenda amb les adreces codificades en el programari, adreces de les que també disposa l'aranya.
E. L'aranya inicia la seva execució amb l'inici de la simulació, això comporta que les primeres connexions no li aportin cap tipus d'informació addicional, ja que els nodes encara no han realitzat el descobriment de veïns.

id	simulationTime	origin	destination	user_agent
1	0.636804185015	33882297	50594591	Crawler
2	1.83210605638	33882297	50725365	Crawler
3	2.34125491453	33882297	51052765	Crawler
4	2.45015319289	33882297	51118302	Crawler
5	3.90381002445	33882297	50332398	Crawler
6	4.58322794099	33882297	50921931	Crawler
7	7.06057032883	33882297	50921691	Crawler
8	7.65855379345	33882297	50987963	Crawler
9	7.84372151725	33882297	50921526	Crawler
10	72.9314799524	33882297	50594591	Crawler
11	72.9314799524	50594591	33882297	reference
12	75.5940436985	33882297	50725365	Crawler
13	75.5940436985	50725365	33882297	reference

Imatge 29: missatges *version* enviats per l'aranya a l'inici de la simulació.

Si analitzem aquests fets arribem a la conclusió que el quid de la qüestió està en els factor E i en el retard en el missatge *DNS Response* que apreciem en la Imatge 27. En aquest sentit, si es soluciona aquest problema el rendiment de l'aranya hauria de millorar.

Connexions creades
Per últim, exposarem les connexions creades, però no es procedirà a analitzar-les ja que no són rellevants en aquest moment pels resultats de l'aranya que acabem de comentar.

Valor	Resultats / Valors		Descripció / Observacions
	Simulació I.A	Simulació I.B	
Connexions de sortida a DNS	179.619	179.642	
Connexions d'entrada provinents de DNS	71.005	71.086	
Connexions establertes entre nodes	3.189	3.190	
Connexions amb l'aranya	8	12	

Taula 30: valors estadístics de les connexions en la simulació.

Conclusions

Recollint tot el que s'ha exposat en aquest capítol, si procedim al càlcul de l'eficàcia de l'aranya és del 0.03 dels nodes existents. El valor s'obté de la mitja dels següents xifres:

$$Eficàcia\ simulació\ I.A = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{17}{899} = 0.02$$

$$Eficàcia\ simulació\ I.B = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{23}{899} = 0.03$$

Tot i que aquest percentatge seria molt positiu per la comunitat Bitcoin si fos extrapolable a Bitnodes (en el sentit que el número de nodes de Bitcoin seria un 0.97 superior al detectat), la xifra presentada és incorrecte per les qüestions indicades anteriorment i, per tant, en cap cas, extrapolable.

Recapitulant, la següent simulació l'executarem amb els següents canvis:

- Ajustar el valors del DNS per tal que únicament l'1% dels nodes siguin servidors DNS.
- Augmentar el valor de les variables SELF_ANNOUNCEMENT i PING_ANNOUNCEMENT perquè s'enviïn menys missatges i millori el rendiment.
- Resoldre el problema del retard en la tramesa de missatges *DNSResponse* tal com hem indicat.

8.3.2 Segona simulació

Per l'execució de la segona simulació s'han realitzat els següents canvis en les constants i variables del simulador:

Variables d'usuari

Atribut	Justificació del canvi	Valor
Tipus de nodes	Per error en la primera execució es va introduir el 10% de DNS.	DNS: 01% Node: 99% Aranya: 1 unitat

Taula 31: configuració d'usuari per la 2a simulació

Variables programades al simulador

Classe	Constant	Justificació del canvi	Valor antic	Valor nou
Node	SELF_ANNOUNCEMENT	S'ha augmentat el temps que un anunci s'anuncia a la xarxa per disminuir el tràfic ja que no aporta informació en la simulació.	300ut	600ut
	PING_ANNOUNCEMENT		175ut	300ut
	Connexions	El node manté les seves connexions durant tota la simulació	Sí	No

Taula 32: configuració programàtica per la 2a simulació.

Addicionalment, també s'ha modificat el comportament del càlcul del retard dels missatges per tal d'adreçar els problemes detectats en la simulació anterior.

Les dades obtingudes en la simulació són les que presentem a continuació.

Dades genèriques

Les dades genèriques de la simulació són les següents:

Valor	Resultats / Valors		Observacions
	Simulació II.A	Simulació II.B	
Característiques de maquinari	2 CPU 30 GB RAM	4 CPU 15 MB RAM	
RAM màxima	-	-	Similars a la primera simulació.
Ús CPU màxim	-	-	
Temps d'execució	1,11 hores	1,18 hores	Execució de la simulació.
Temps de post-processament	< 6 hores	< 6 hores	En el simulador no es calcula quan es tarda a executar les simulacions.
Mida de la BD	59 MB	57,6 MB	Mida de la BD (en format de text).

Taula 33: dades genèriques de la 2a simulació

Respecte a la primera simulació es pot veure com el temps d'execució s'ha reduït en un 80% (de 9 hores a 1.2 hores). Aquest fet es pot deure a:

- La reducció en el número de missatges *addr* i *ping*.
- El canvi en el mètode de càlcul de la mida dels missatges.

Així mateix també s'ha reduït el temps de post-processament de les dades i la mida que ocupa la base de dades.

Dades dels nodes

Valor	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
Número de nodes	1000		
Número *nodes convencionals*	989		
Número de DNS	10		
Número d'aranyes	1		
Bootstrap			
Ús de DNS?	Sí		S'han usat els servidors DNS per arrancar?
Número d'adreces per DNS	~205		Número mig d'adreces per servidor DNS (valor aproximat)
Nodes que els han emprat	989		Número de nodes que han emprat l'arrancada per DNS
Ús d'adreces codificades?	No		A excepció de l'aranya
Número d'adreces codificades	~90		Número d'adreces codificades en els nodes
Nodes que les han emprat	únicament l'aranya		

Taula 34: dades dels nodes per la 2a simulació.

Amb els canvis introduïts en el mètode de càlcul de de la mida dels missatges podem veure com els nodes (a excepció de l'aranya) arranquen la xarxa mitjançant els DNS en lloc d'emprar les adreces codificades, que és el comportament esperat.

D'altre banda, el número de nodes coincideix amb l'especificat.

Dades dels missatges

Valor	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
Total de missatges	86.730	81.452	Número de missatges enviats en el simulador. Únicament es contemplen els missatges relacionats amb la xarxa. La base de dades recull dos cops el missatge: quan s'envia i quan es rep, aquestes dades corresponen a un dels dos esdeveniments.
msg. *addr*.	32.522	27.887	
msg. *close*.	1.162	1.304	
msg. *DNS none*	0	0	
msg. *DNS query*	9.890		
msg. *DNS response*	9.889		
msg. *GetAddr*	3.440	3.563	
msg. *Reject*	15.625	14.278	
msg. *VerAck*	3.440	3.563	

Valor	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
msg. *Version*	10.761	11.078	

Taula 35: dades dels missatges per la 2a simulació.

Respecte a la primera simulació observem les següents diferències:

- Podem veure com l'augment de la variable SELF_ANNOUNCEMENT de 350 a 600 unitats de temps ha disminuït exponencialment el número de missatges *addr* enviats.
- També podem veure com el número de missatges relacionats amb els DNS han disminuït al haver-se reduït el número de clients DNS en el sistema.
- Així mateix també s'han reduït el número de missatges *version* i *reject* com a conseqüència directa de la inicialització dels nodes per DNS. Al disposar tots els nodes de més informació sobre la xarxa (que no sigui idèntica) les connexions que es realitzen estan més distribuïdes, pel que augmenten en número i es redueixen el número de missatges de rebuig.

Pel que fa el motiu del missatge *Reject,* el 100% dels missatges es deu a "Connection not established"; el que implica que un node rep missatges abans d'establir una connexió.

Dades de l'aranya

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
Missatges enviats			
Núm. enviats *close*.	10	9	
Núm. enviats *GetAddr*	8	7	
Núm. enviats *Reject*	13	1	
Núm. enviats *VerAck*	8	7	
Núm. enviats *Version*	9	9	
Missatges rebuts			
Núm. rebuts *Addr*.	10	7	
Mitjana d'adreces rebudes per *Addr*	1	76,7	
Número d'adreces rebudes diferents	9	399	
Cops que l'aranya rep la seva adreça	0	6	
Núm. rebuts *close*.	0	1	
Núm. rebuts *GetAddr*	0	0	
Núm. rebuts *Reject*	2	4	
Núm. rebuts *VerAck*	0	0	

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
Núm. rebuts *Version*	10	12	
Informació sobre els nodes			
Nodes coneguts per l'aranya	104	368	Incloent les adreces codificades i descartant les direccions duplicades i l'adreça de l'aranya.
Adreces codificades en l'aranya	95	90	

Podem observar com ha millorat el coneixement de la xarxa per part de l'aranya, tot i que seguim observant que l'aranya realitza poques connexions, el que creiem que es pot deure a que el número màxim de connexions està establert en 8. Per tant, en la propera simulació optarem per augmentar-lo.

Connexions creades

Valor	Resultats / Valors		Descripció / Observacions
	Simulació II.A	Simulació II.B	
Connexions de sortida a DNS	9.890		
Connexions d'entrada provinents de DNS	9.305	9,291	
Connexions establertes entre nodes	3.069	3.159	
Connexions amb l'aranya	8	7	

Conclusions

Recollint tot el que s'ha exposat en aquest capítol, si procedim al càlcul de l'eficàcia de l'aranya és del 0.28 dels nodes existents. El valor s'obté de la mitja dels següents xifres:

$$Eficàcia\ simulació\ II.A = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{104}{989} = 0.11$$

$$Eficàcia\ simulació\ II.B = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{459}{989} = 0.46$$

Tot i que el número de connexions i l'eficàcia de l'aranya ha millorat respecte a la simulació anterior, cal destacar el baix número de connexions i d'intents de connexió de l'aranya. En aquest sentit, en la propera simulació augmentarem el número màxim de connexions per tal de comprovar si amb aquest canvi millora significativament l'eficàcia de l'aranya.

8.3.3 Tercera simulació

En la tercera simulació s'han dut a terme els següents canvis:

Classe	Constant	Justificació del canvi	Valor antic	Valor nou
Node	MAX_CONNECTIONS	Comprovar l'afectació a l'aranya	8	15

Taula 36: canvi en els valors de la configuració en la 3a simulació.

Que han generat les següents dades:

Dades genèriques

Valor	Resultats / Valors		Observacions
	Simulació III.A	Simulació III.B	
Característiques de maquinari	2 CPU 30 GB RAM	4 CPU 15 MB RAM	
RAM màxima	-	-	Similars a la primera simulació.
Ús CPU màxim	-	-	
Temps d'execució	1,39 h	1,58 h	Execució de la simulació.
Temps de post-processament	-	-	En el simulador no es calcula quan es tarda a executar les simulacions.
Mida de la BD	64,9 MB	70,2 MB	Mida de la BD (en format de text).

Taula 37: dades generals de la 3a simulació

Podem veure com ha augmentat lleugerament el temps d'execució, el que és plenament comprensible si es considera que s'han enviat més missatges a l'ampliar-se el número màxim de connexions.

Dades dels nodes

Valor	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Número de nodes	1.000	1.000	
Número *nodes convencionals*	989	989	
Número de DNS	10	10	
Número d'aranyes	1	1	
Bootstrap			
Ús de DNS?	Sí	Sí	S'han usat els servidors DNS per arrancar?
Número d'adreces per DNS	~205	~205	Número mig d'adreces per servidor DNS (valor aproximat)
Nodes que els han emprat	989	989	Número de nodes que han emprat l'arrancada per DNS
Ús d'adreces codificades?	No	No	A excepció de l'aranya

Valor	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Número d'adreces codificades	93	95	Número d'adreces codificades en els nodes
Nodes que les han emprat	L'aranya és l'únic node que ha emprat l'arrancada		

Taula 38: dades dels nodes en la 3a simulació

Les dades són similars a les de la simulació anterior, pel que no les comentarem.

Dades dels missatges

Valor	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Total de missatges	131.704	177.369	Número de missatges enviats en el simulador. Únicament es contemplen els missatges relacionats amb la xarxa. La base de dades recull dos cops el missatge: quan s'envia i quan es rep, aquestes dades corresponen a un dels dos esdeveniments.
msg. *addr*.	59.445	91.538	
msg. *close*.	747	890	
msg. *DNS none*	0	0	
msg. *DNS query*	9.890	9.890	
msg. *DNS response*	9.888	9.883	
msg. *GetAddr*	4.528	4.684	
msg. *Reject*	29.150	41.806	
msg. *VerAck*	4.529	4.684	
msg. *Version*	13.527	13.994	

Taula 39:dades dels missatges en la 3a simulació

Podem observar com l'augment del número mínim de connexions ha augmentat lleugerament els missatges tramesos en la simulació.

Dades de l'aranya

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Missatges enviats			
Núm. enviats *close*.	16	16	
Núm. enviats *GetAddr*	13	12	
Núm. enviats *Reject*	19	21	
Núm. enviats *VerAck*	13	12	
Núm. enviats *Version*	16	16	

Tipus de missatge	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Missatges rebuts			
Núm. rebuts *Addr*.	29	31	
Mitja d'adreces rebudes per *Addr*	24,03	28,19	
Número d'adreces rebudes diferents	490	562	
Cops que l'aranya rep la seva adreça	8	10	
Núm. rebuts *close*.	2	0	
Núm. rebuts *GetAddr*	0	0	
Núm. rebuts *Reject*	12	9	
Núm. rebuts *VerAck*	0	0	
Núm. rebuts *Version*	15	16	
Informació sobre els nodes			
Nodes coneguts per l'aranya	489	561	Incloent les adreces codificades i descartant les direccions duplicades i l'adreça de l'aranya.
Adreces codificades en l'aranya	93	95	

Taula 40: dades de l'aranya en la 3a simulació.

Com podem veure, en relació a les simulacions anteriors, ha augmentat lleugerament el número de connexions establertes, però segueix existent un baix número d'intents de connexions i molt pocs missatges d'inici de connexió enviats.

En conseqüència considerem que tot i que es segueixi augmentant el màxim de connexions, l'aranya tindrà un comportament similar, pel que concloem que s'hauria de procedir a analitzar més exhaustivament l'algorisme que empra l'aranya per determinar-ne el rendiment.

Connexions creades

Valor	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Connexions de sortida a DNS	19.778	19.773	
Connexions d'entrada provinents de DNS	9.275	9.284	
Connexions establertes entre nodes	8.052	8.304	

Valor	Resultats / Valors		Descripció / Observacions
	Simulació III.A	Simulació III.B	
Connexions amb l'aranya	11	12	

Taula 41: dades de les connexions en la tercera simulació

Conclusions

Recollint tot el que s'ha exposat en aquest capítol, si procedim al càlcul de l'eficàcia de l'aranya, aquesta detecta el 0.52 dels nodes existents. El valor s'obté de la mitja dels següents xifres:

$$Eficàcia\ simulació\ III.A = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{489}{989} = 0.49$$

$$Eficàcia\ simulació\ III.B = \frac{Número\ nodes\ descoberts}{Número\ total\ de\ nodes} = \frac{561}{989} = 0.57$$

Els valors són relativament similars als de la simulació anterior, pel que considerem que si seguim ampliant el número màxim de connexions no obtindrem una millora rellevant de l'eficàcia de l'aranya.

En conclusió, som partidaris d'establir la configuració emprada en aquesta simulació com a base de l'eina BTC-NS i assumir que, amb l'algorisme actual, l'aranya podrà detectar aproximadament el 52% dels nodes.

9 Millores futures

En aquest capítol procedirem a enumerar les millores que en un futur es podrien implementar en el simulador, presentant-ne els motius i avantatges, però sense entrar en detall en el seu desenvolupament. Es poden dividir les millores en dos grans grups: aquelles que impliquen una millora del rendiment i les que permetrien millorar l'estructuració i escalabilitat del sistema.

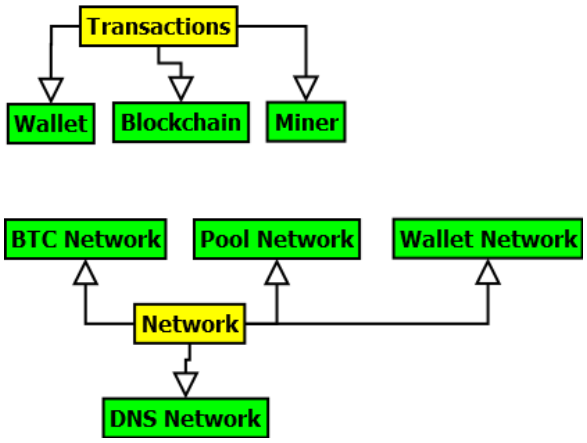
Siguin del tipus que siguin, considerem que les millores aquí plantejades podrien ser millorar el simulador, facilitant en un futur les simulacions.

9.1 Arquitectura dels nodes

En l'apartat 6.2 mostràvem com existeix una classe *Node* que implementa la majoria de característiques d'un client en el simulador i que són heretades per tots els tipus de nodes (tal com es mostra en la Imatge 19). Això comporta que qualsevol node, independentment del tipus, disposarà pràcticament totes les característiques d'un client complet.

Per tant, si un tipus de node, com l'aranya, no ha de disposar de determinades característiques, aquestes s'han d'eliminar explícitament del node (ja sigui invocant la comanda *del* o bé sobreescrivint totes les funcions afectades). Com es pot veure aquest sistema és propens a l'error, pel que s'hauria d'evitar si és possible.

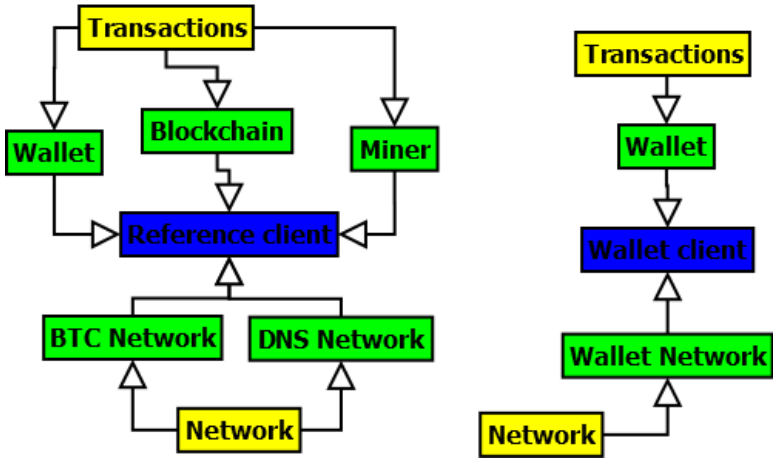
Per aquest motiu, es proposa una reestructuració de l'arquitectura de classes que implementen les funcionalitats dels nodes. En particular, en lloc d'implementar les característiques en un classe corresponent a un tipus de node, es proposa implementar les classes d'acord a la seva funcionalitat agrupant-les tal com es mostra en la següent imatge[60]:



Imatge 30: arquitectura de classes proposada com a millora

Amb aquest canvi els nodes es construirien d'acord a les funcionalitats que necessitessin. Per exemple, per construir un client de referència, s'utilitzarien totes les característiques representades en verd (excepte "*Wallet Network*" i "*Pool Network*"); i, per construir un client moneder únicament serien necessàries les funcionalitats "*Wallet*" i "*Wallet Network*".

[60] El diagrama representa l'arquitectura general i no es defineixen exactament tots els elements necessaris (que es podrien agrupar en classes o seguir una programació orientada a aspectes).



Imatge 31: exemple de dependències d'un client complet i d'un client moneder.

Aquesta millora podria comportar les següents avantatges respecte a la implementació actual:

- Facilitaria la definició dels diferents tipus de nodes existents.
- Disminuirien lleugerament els requisits de memòria, ja que els nodes únicament implementarien les seves necessitats.
- Facilitaria la implementació de diverses versions d'una mateixa funcionalitat, augmentant l'escalabilitat del sistema. El que, a la vegada, ajudaria a la implementació de diversos "perfils de comportament" dels nodes"¡ (per exemple, diversos motors d'exploració per l'aranya o facilitar la implementació de nous clients en el simulador del mateix tipus).
- Es podria reaprofitar la majoria del codi existent, tot i que s'hauria d'adaptar.
- Asseguraria que els tipus de nodes especials (l'aranya i el servidor DNS) no implementessin funcionalitats que no fossin necessàries.
- Les diferents funcionalitats es podrien personalitzar per adaptar-les a un mode concret del simulador. Per exemple, la *BTC Network* quan el simulador està configurat amb una xarxa estàtica podrà ser diferent de quan està configurat per treballar amb una xarxa completament dinàmica (en el primer cas, per exemple, no serà necessària una agenda d'adreces).
- Ajudaria a implementar les diferents versions del protocol Bitcoin.

9.2 Millores en la definició de la xarxa

A dia d'avui les propietats de la xarxa (latència i ample de banda) es defineixen per cada connexió de forma independent. Aquest fet comporta tres inconvenients:

- Per cada connexió s'han de definir els dos atributs el que comporta una inversió en memòria.
- Pel mode de "descobriment de veïns" pot succeir que dos nodes es connectin, tanquin la seva connexió i es tornin a connectar. En aquesta situació ens podem trobar que les dues connexions tinguin uns valors de xarxa diferents.
- Pel mode de "descobriment de veïns", si s'utilitza la definició avançada de la xarxa (on mitjançant un fitxer s'especifiquen les condicions de la xarxa) caldria definir totes les possibles connexions entre els nodes. Això comporta que caldria carregar totes les dades a memòria (o definir una nova base de dades, temporal, on emmagatzemar aquesta informació per obtenir-la quan es necessités).

Per aquests motius es proposa canviar el mètode com s'especifica la xarxa per tal que la informació sobre la xarxa sigui un atribut més del node. D'aquesta forma, es definirien

uns valors d'ample de banda i latència màxims per cada node i, per cada ús de la xarxa, es determinarien dinàmicament:

- L'ample de banda d'una connexió serà el mínim dels dos iguals que hi participen.
- La latència serà la de l'emissor del missatge, o la suma de la latència dels dos nodes.
- Addicionalment es podria considerar que l'ample de banda d'un node es podria dividir pel número de connexions (com més connexions tingui un node, menys ample de banda ja que ha de dividir la seva disponibilitat).

Amb aquest sistema considerem que s'aconseguiria combatre els tres desavantatges descrits anteriorment.

9.3 Millores en el rendiment

En el descobriment de veïns (sobretot en el mètode *Node.periodic_network*) es realitzen diverses iteracions sobre les connexions de cada node. Aquestes iteracions, especialment en simulacions amb un alt volum de nodes, comportaran un problema de rendiment que es podria combatre si s'agrupen les diverses iteracions o s'empren índexs que facilitin la tasca.

Per exemple, per determinar amb quins nodes s'ha establert una connexió, actualment s'executa el següent codi:

```
def get_established_connections(self):
    """
    :return: a list with the established connections (Link.status ==
    ESTABLISHED or DONE.
    :rtype:list[Link]
    """
    result = list()
    for link in self.links.values():
        if link.status in (LinkStatus.ESTABLISHED, LinkStatus.DONE):
            result.append(link)
    return result
```

Codi 37: iteració per obtenir les connexions establertes

En lloc d'invocar bucles diversos cops, es podrien emmagatzemar les connexions en un diccionari com per exemple el següent:

```
links = {LinkStatus.NONE: list(),
         LinkStatus.VERSION: list(),
         LinkStatus.ESTABLISHED: list(),
         LinkStatus.DONE: list(),
         LinkStatus.DNS: list()}
```

Codi 38: suggeriment per la millora de la gestió de les connexions.

9.4 Millores en el rendiment de la base de dades

Cada cop que s'ha d'emmagatzemar un esdeveniment es realitza una crida a base de dades de forma individual, el que comporta una gran ineficiència al dedicar-se prop d'un terç del temps d'execució de la simulació únicament a base de dades[61]

[61] D'acord a les xifres calculades en el capítol 8.

Per accelerar el procés, s'han identificat tres formes que poden millorar la gestió de la base de dades:

I. Establir un sistema intermediari que agrupi les diverses insercions individuals en la base de dades i les trameti conjuntament tal com es proposa en l'apartat 9.4 [23].
II. Procurar evitar guardar dos cops el mateix missatge tal com es proposa en l'apartat 9.5.
III. Realitzar una administració de la base de dades MySql per tal d'intentar millorar-ne la velocitat.

9.5 Millores en la gestió dels missatges

En la implementació actual del simulador, cada cop que s'envia un missatge s'emmagatzema dos cops: quan s'envia i quan el rep el destinatari de la comunicació, pel que es realitzen dos crides d'inserció. Es podria millorar el rendiment si únicament es guardés en base de dades el missatge un únic cop: en el moment de rebre'l. Per aconseguir-ho caldria:

A. Enviar amb el missatge les dades del node emissor que es vulguin guardar en base de dades com, per exemple, l'id d'origen o el moment en que s'ha tramés.
B. Al rebre el missatge, s'hi hauria d'incloure el moment en que s'han rebut abans de guardar-lo en base de dades.

Amb aquesta proposta es reduirien a la meitat els missatges emmagatzemats, millorant el rendiment, a la vegada que es considera que es mantindria tota la informació necessària.

D'altre banda es podria aprofitar aquesta modificació per incloure en les dades que es guarden en base de dades la següent informació:

- Si al rebre un missatge el node el rebutja (per exemple perquè no s'ha establert una connexió).
- El retard en la transmissió (tot i que no seria estrictament necessari ja que es pot calcular amb la diferència entre el moment de recepció i el de transmissió.
- En quin punt del codi o per quin motiu s'han generat. Això permetria detectar quan s'envia un missatge i, per tant, aprofundir en un estudi o ajustar el comportament del simulador. La motivació d'aquesta millora s'exposa en l'apartat 0.

9.6 Especificar els tipus de nodes numèricament

Actualment el número de nodes s'indica en el simulador mitjançant un percentatge, independentment del seu tipus. Seria convenient, per tal d'afegir més flexibilitat al simulador, permetre la definició dels tipus de node numèricament. Això és d'especial interès pels nodes "especials" (DNS, aranyes) ja que, en condicions normals, al contrari que amb altres nodes, se'n necessitarien poques instàncies.

9.7 Implementar un mecanisme de nivell de servei

Per tal d'emular el comportament dels nodes en el món real, caldria que en ocasions es comportin de forma erràtica: per exemple desconnectant-se durant períodes de temps aleatoris. Aquesta funcionalitat és important si es vol emprar el BTC-NS com a medi per l'estudi de la xarxa d'iguals que sosté Bitcoin (com per exemple en el present projecte).

En aquest sentit, en el codi desenvolupat en el marc del projecte s'implementa un mecanisme per tal que el node deixi d'estar actiu, aleatòriament, durant períodes compresos entre 10 i 100 unitats de temps en el simulador. Tot i que creiem que el

comportament desenvolupat en el marc del projecte proporciona una base per l'estudi de les propietats de la xarxa, pensem que es pot millorar mitjançant un sistema de "nivell de servei".

Tradicionalment en el món de l'administració de sistemes, es defineix com a "nivell de servei" el temps (normalment mesurat amb un percentatge) que està disponible un servei tecnològic. I és precisament en aquest sentit el sistema que es recomana implementar: desenvolupar un mecanisme que permeti establir amb quina probabilitat un node es desconnectarà de la xarxa i durant quant de temps no estarà disponible.

Aquest sistema també es podria emprar per seleccionar els nodes amb millor "nivell de servei" (i, per tant, de millor reputació) per tal que les seves adreces estiguin codificades en els clients o els servidors DNS.

9.8 Ajustar les "unitats de temps" del simulador

La principal diferència entre Bitcoin Core, "el client de referència", i el simulador és la gestió del temps: el primer funciona en temps real –il·limitat– mentre que el segon treballa amb temps discrets. Per tant, per tal d'emular el comportament caldrà realitzar diversos anàlisis (el que es realitza en aquest estudi en serà un) que permetin ajustar el comportament de les constants del BTC-NS.

9.9 Millores diverses

Per últim, i concloent aquest apartat, hem confeccionat una llista de les tasques pendents d'implementar en el simulador, algunes de les quals figuren com a "comentaris TODO" en el propi codi.

- Implementar la configuració avançada del simulador en el mode "xarxa dinàmica".
- Limitar l'ús del node DNS a la "xarxa dinàmica", ja que no té sentit en els altres mètodes de funcionament del simulador.
- En el mode "xarxa dinàmica", el DNS disposarà com a màxim del 25% de les adreces d'una simulació. Aquest percentatge s'hauria d'ajustar d'acord al número de nodes de la simulació (com més nodes, el percentatge hauria de ser més baix).
- Implementar les versions més rellevants del protocol Bitcoin, gestionant un sistema d'incompatibilitats.
- Afegir al simulador l'opció d'executar-se mitjançant línia de comandes.
- Implementar un sistema que permeti rebre avisos (per correu electrònic per exemple) quan s'hagi finalitzat una simulació.
- Millorar el temps de "post processament" de la simulació, si és possible.
- Impedir el comportament descrit en l'apartat 7.3.1.1.

10 Conclusions i treball futur

Per finalitzar aquest document, plantejarem les conclusions i el treball futur que es pot dur a terme a partir de les tasques desenvolupades. En concret, en primer lloc recuperarem del capítol 1.2 els objectius i resultats del projecte i procedirem a avaluar-ne l'acompliment:

L'assoliment dels objectius del projecte és:

Codi	Descripció	Assoliment
O1	Redactar una introducció a Bitcoin.	Realitzat en el capítol 4.
O2	Analitzar el funcionament de Bitcoin a nivell de xarxa.	Realitzat en el capítol 5.
O3	Documentar el funcionament del BTC-NS, detectant els punts que divergeixen substancialment de la xarxa real.	Realitzat en els capítols 6.
O4	Implementar les millores detectades en l'O3.	Realitzat en el capítol 7.
O5	Estudiar el comportament de l'aranya.	Aconseguit en el capítol 8.

Taula 42: objectius assolits en el projecte.

I pel que fa als resultats esperats:

Codi	Descripció	Assoliment
R1	Documentació d'introducció a les tecnologies Bitcoin.	Realitzat.
R2	Descripció del funcionament de la xarxa de d'iguals	Realitzat.
R3	Anàlisi del funcionament del simulador a nivell de xarxa.	Realitzat.
R4	Resultats de l'estudi del comportament de l'aranya.	Realitzat
R5	Memòria final del projecte.	Realitzat.

Taula 43: resultats assolits en el projecte.

Per tant, considerem que hem dut a terme tots els objectius i resultats planificats tot i que, en un futur caldrà analitzar els problemes detectats en el funcionament de l'aranya que, en els millors dels casos estudiats, li permet analitzar la tipologia de la xarxa amb un 0.52 de precisió (objectiu O4 i resultat R4).

Per tant, com que considerem que l'entorn de simulació és prou estable, el següent pas que duríem a terme per millorar l'eficàcia de l'aranya seria procedir a analitzar el seu comportament i detectar quins aspectes s'han de canviar.

Per últim, exposarem que, com a conseqüència directa del projecte, s'ha ajustat el funcionament de la xarxa de parells del simulador per tal que segueixi el funcionament *estàndard*, el que servirà com a base per dur a terme simulacions que permetin analitzar el comportament de la xarxa d'iguals Bitcoin.

10.1 Treball futur

En un altre ordre de coses, pel que fa el treball que es podrà desenvolupar més endavant, d'una banda s'hauran d'implementar –consensuant la decisió amb la resta de desenvolupadors del simulador– les millores i problemes identificats en el capítol 9.

Un cop adreçades aquestes qüestions considerem que, pel que fa a la xarxa de parells, el simulador es podria arribar a publicar a Internet per al seu ús (ja sigui el codi font o una plataforma on realitzar les simulacions, si es troba el finançament).

Per últim, les eines de programari desenvolupades al llarg del projecte podrien arribar a usar-se a llarg termini per l'anàlisi del comportament d'algorismes de xarxa, ja sigui realitzant anàlisis directes simulats o duent a terme simulacions per estimar el grau d'error d'estudis en el món real.

Referències

[1] M. Himsolt, «GML, Graphic Modeling Language,» [En línia]. Disponible: http://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf. [Últim accés: 21 08 2015].

[2] Bitcoin Project, [En línia]. Available: https://bitcoin.org/es/descargar. [Últim accés: 17 08 2015].

[3] A. Yeow, «Bitnodes is currently being developed to estimate the size of the Bitcoin network by finding all the reachable nodes in the network.,» [En línia]. Disponible: https://getaddr.bitnodes.io/. [Últim accés: 20 08 2015].

[4] BlockChain.info, [En línia]. Available: https://blockchain.info/es/connected-nodes. [Últim accés: 20 08 2015].

[5] A. Biryukov, D. Khovratovich i I. Pustogarov, «Deanonymisation of clients in Bitcoin P2P network,» *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security,* 2014.

[6] J. A. Donet, C. Pérez i J. Herrera, «The Bitcoin P2P network,» [En línia]. Disponible: http://www.deic.uab.cat/~jherrera/fitxers/FC2014-donet-perez-herrera.pdf. [Últim accés: 21 08 2015].

[7] A. Miller, J. Litton, A. Pachulski, N. Gupta, N. Spring i B. Bhattacharjee, «Discovering Bitcoin's Public Topology and Influential Nodes,» [En línia]. Disponible: http://cs.umd.edu/projects/coinscope/coinscope.pdf. [Últim accés: 23 08 2015].

[8] Q. Lv, P. Cao, E. Cohen, K. Li i S. Shenker, «Search and replication in unstructured peer-to-peer networks,» de *ICS '02 Proceedings of the 16th international conference on Supercomputing*, New York, ACM, 2002, pp. 84-95.

[9] M. Schlosser, T. E. Condie i S. D. Kamvar, «Simulating a File-Sharing P2P Network,» de *1st Workshop on Semantics in Grid and P2P Networks*, 2003.

[10] S. Nakamoto, «Bitcoin: A Peer-to-Peer Electronic Cash System,» [En línia]. Disponible: https://bitcoin.org/bitcoin.pdf. [Últim accés: 30 08 2015].

[11] A. Narayanan, J. Bonneau, E. W. Felten i A. Miller, «Bitcoin and Cryptocurrency Technologies,» Coursera, 2015.

[12] A. M. Antonopoulos, Mastering Bitcoin, O'Reilly Media, Inc., 2014.

[13] M. Miller, The Ultimate Guide to Bitcoin, Que, 2014.

[14] M. Swan, Blockchain, O'Reilly Media, Inc, 2015.

[15] Ethereum Switzerland GmbH, «Ethereum Frontier,» [En línia]. Disponible: https://ethereum.org/. [Últim accés: 07 09 2015].

[16] S. Nakamoto i B. C. developers, «Bitcoin Core,» [En línia]. Disponible: https://github.com/bitcoin/bitcoin. [Últim accés: 10 10 2015].

[17] P. Wuille i G. Andersen, «CAddrMan: stochastic address manager #787,» 21 01 2012. [En línia]. Disponible: https://github.com/bitcoin/bitcoin/pull/787. [Últim accés: 11 10 2015].

[18] C. Pérez-Solà i J. Herrera-Joancomartí, «Development of an open source bitcoin network simulator,» Barcelona, 2014.

[19] Simpy developers, «Simpy's Shared Resources: Stores,» [En línia]. Disponible: https://simpy.readthedocs.org/en/latest/topical_guides/resources.html#res-type-store. [Últim accés: 24 10 2015].

[20] Team SimPy, «Simpy: event discrete simulation for Python.,» [En línia]. Disponible: https://simpy.readthedocs.org/en/latest/. [Últim accés: 12 10 2015].

[21] Python Software Foundation, «25.3. unittest — Unit testing framework,» [En línia]. Disponible: https://docs.python.org/2/library/unittest.html. [Últim accés: 31 10 2015].

[22] Python Software Foundation, «PEP 0008 -- Style Guide for Python Code,» 15 07

2001. [En línia]. Disponible: https://www.python.org/dev/peps/pep-0008/. [Últim accés: 31 10 2015].

[23] Oracle Corporation, «MySQL 5.1 Reference Manual, Speed of INSERT Statements,» [En línia]. Disponible: https://dev.mysql.com/doc/refman/5.1/en/insert-speed.html. [Últim accés: 01 11 2015].

[24] Bitcoin Wiki, «Transaction,» [En línia]. Disponible: https://en.bitcoin.it/wiki/Transaction. [Últim accés: 06 09 2015].

[25] I. Grigg, «Bitcoin and the Byzantine Generals Problem -- a Crusade is needed? A Revolution?,» 19 11 2014. [En línia]. Disponible: http://financialcryptography.com/mt/archives/001522.html.

[26] P. Wuille, «[Bitcoin-development] CAddrMan: Stochastic IP address manager,» 30 01 2012. [En línea]. Disponible: http://lists.linuxfoundation.org/pipermail/bitcoin-dev/2012-January/001100.html. [Último acceso: 11 10 2015].

[27] Wikipedia, the free encyclopedia, «Web crawler,» [En línia]. Disponible: https://en.wikipedia.org/wiki/Web_crawler. [Últim accés: 26 08 2015].

[28] B. Foundation, «Bitcoin developer's guide,» [En línia]. Disponible: https://bitcoin.org/en/developer-guide#p2p-network. [Últim accés: 05 12 2015].

[29] Bitcoin Wiki contributors, «Protocol documentation,» [En línia]. Disponible: https://en.bitcoin.it/wiki/Protocol_documentation. [Últim accés: 18 10 2015].

Annexos

Annex I: tipus de missatges del protocol de Bitcoin

Els missatges disponibles en el protocol Bitcoin són els que es mostren a continuació. La primera columna, "Tipus", indica la denominació del missatge; la segona, "Contingut", què conté i la columna "Descripció" una petita explicació.

Tipus	Contingut	Descripció
Version	La versió del protocol.	Missatges d'inicialització de la connexió. Veure 5.1 per més informació.
VerAck	N/A	Confirmació del missatge version (*Version Ack.*)
Addr	Màxim 1.000 adreces.	Envia direccions dels nodes coneguts.
Inv	Màxim 50.000 entrades o 1.8MiB	Comunica, de forma sol·licitada o no, informació que el node coneix.
Getdata		Petició per obtenir continguts. Es sol respondre amb un missatge *Inv*.
Notfound		Resposta a Getdata si no es disposa dels continguts sol·licitats
Getblocks		Petició per obtenir blocs. La resposta és un missatge Inv amb els blocs.
Getheaders		Petició per obtenir les capçaleres dels blocs després d'un bloc concret.
Tx	Una transacció	Tramet una transacció que no estigui en un bloc en resposta a un missatge *getdata*.
Block	Un bloc	Tramet un bloc. Generalment com a resposta a un missatge *getdata*.
Headers	Màxim 1.8MiB	Tramet les capçaleres com a resposta a un missatge *getheaders*.
Getaddr	N/A	Petició per obtenir adreces sobre nodes.
Mempool	Un filtre[62].	Sol·licita informació sobre una transacció
Ping[63]	Número aleatori	Per confirmar que la connexió Bitcoin segueix establerta.
Pong	Número aleatori del ping	Resposta a un missatge ping.
Reject	Codi i descripció de l'error.	Missatge que retorna un codi d'error.
Filterload Filteradd Filterclear Merkleblock	Descripció d'un filtre.	Missatges descrivint els filtres especificats en el missatge *mempool*.
Alert	Informació sobre error.	Missatge notificant un error en el protocol Bitcoin.

Taula 44: tipus de missatges de Bitcoin

[62] Per millorar la privacitat el missatge no demana una transacció en concret, envia un filtre i l'igual que rebi el missatge retornarà totes les transaccions que compleixin dit filtre.
[63] No s'ha de confondre amb l'eina *ping* que treballa amb missatges d'ICMP a nivell d'IP.

Annex II: tipus de nodes de Bitcoin

Podem dividir en dos grans tipologies els nodes que formen part de la xarxa: els que participen directament en Bitcoin, ja sigui generant blocs o reenviant transaccions, i aquells que no hi participen directament, com per exemple les aranyes o els nodes tipus client.

Entre els primers hi podem trobar:

Nom node	Característiques			
	Còpia cadena de blocs	Funcionalitats de xarxa	Sistema mineria	Moneder
Client de referència (Bitcoin Core)	Sí	Sí	Sí	Sí
Node complet (Full):	Sí	Sí		
Miner	Sí	Sí	Sí	
Moneder lleuger (*Lightweight wallet*)		Sí		Sí
Portes d'enllaç (*Gateways* o *Pool Protocol Servers*)	Opcional	Sí		

Taula 45: tipus de nodes que interactuen directament en la xarxa Bitcoin

Les característiques de la taula anterior són les següents:

- Còpia de la cadena de blocs: el node conserva una còpia completa de la cadena de blocs.
- Funcionalitats de xarxa: el node pot accedir a la xarxa per obtenir, generar o retransmetre transaccions o blocs.
- Sistema de mineria: el node actua com un miner, pel que intenta generar blocs que s'incorporin en la cadena de blocs i obtenir Bitcoins en el procés.
- Moneder: el node permet emmagatzemar claus privades i adreces de Bitcoin, pel que podrà gastar Bitcoins.

I entre els segons, els que no hi participen directament, figuren:

Nom node	Característiques			
	Còpia cadena de blocs	Funcionalitats de xarxa	Sistema mineria	Moneder
Aranyes (crawlers)		Sí[64]		
Moneder lleuger stratum		No[65]		Sí

[64] No obstant, la seva interacció en la xarxa es limita a l'obtenció d'informació.
[65] El moneder no té accés directe a la xarxa. Accedeix a ella mitjançant un servidor que actua com a porta d'enllaç.

Nom node	Característiques			
	Còpia cadena de blocs	Funcionalitats de xarxa	Sistema mineria	Moneder
Miner descentralitzat (o *mining pool*)		No[66]	Sí	
Miner coordinador (0 *mining pool leader*)	Sí	Sí[67]		
DNS		Sí[68]		

Taula 46: tipus de nodes que no interactuen directament en la xarxa

[66] El miner forma part d'un sistema distribuït de mineria on realitza les tasques indicades per un servidor central que actua com a porta d'enllaç i coordinador d'aquesta xarxa distribuïda privada.
[67] El coordinador disposa de una doble connexió de xarxa: està connectat a diversos *miners descentralitzats*, mitjançant una xarxa distribuïda privada; i, a la vegada, a la xarxa Bitcoin.
[68] No interactuen directament amb la xarxa. Únicament proporcionen adreces d'altres nodes mitjançant el protocol DNS.

Annex III: arquitectura implementada per la xarxa en Bitcoin Core

En aquest annex intentarem exposar breument l'arquitectura de programari implementada en el client Bitcoin Core per treballar en la xarxa d'iguals. Primer de tot cal considerar que el client implementa una xarxa P2P, pel que rebrà un alt volum de missatges (incloent duplicats) que haurà de redistribuir entre altres membres de la xarxa.

Per tant, un programa convencional, en el benentès de síncron, resultaria totalment ineficaç ja que es trobaria bloquejat per l'alt volum de missatges (cada cop que rebés un missatge es pararia per atendre'l). Per combatre-ho, el programari implementa un conjunt de mètodes asíncrons incloent cues de missatges, semàfors i fils d'execució periòdica.

En aquest sentit, el client disposa d'una estructura que representa a un igual al que està connectat (l'estructura *CNode* analitzada en l'apartat 5.6) i, aquesta classe, té diverses cues i marcadors on el programa va dipositant tant la informació que va rebent del node com la que li ha d'enviar.

Paral·lelament, els fils d'execució asíncrona (que també hem vist a l'apartat 5.6), s'encarreguen de comprovar periòdicament aquestes cues i marcadors per executar determinades accions (com per exemple enviar un missatge amb un determinat contingut) o omplir determinades cues com a conseqüència d'un missatge rebut.

Els dos sistemes es coordinen mitjançant l'ús de semàfors; de forma que, quan un procés està treballant sobre un recurs, un altre procés no hi pot accedir. Amb aquest mecanisme s'evita l'accés concurrent a les mateixes variables i, conseqüentment, evita la corrupció dels recursos compartits entre diferents fils.

Per veure més clarament l'arquitectura de xarxa del programa, a continuació l'exposarem sobre un exemple concret: els missatges de *ping*. Periòdicament, per mantenir viva una connexió els nodes s'intercanvien missatges *ping* i, quan un node el rep, contestarà amb un missatge *pong*. Suposem que el node *A* envia un missatge *ping* a *B*. El comportament de B serà el següent:

I. *A* i *B*, executant els dos el client de Bitcoin Core, estableixen una connexió (per a l'exemple ens és indiferent qui dels dos l'estableix).

II. A l'establir la connexió, *B* crearà una nova instància de la classe *Cnode* per representar a *A.* Aquesta classe contindrà, entre d'altres, l'adreça de l'igual i un conjunt de cues i marcadors.

III. El node *A* envia el missatge ping al node *B*.

IV. Periòdicament, *B* s'executarà el fil *ThreadSocketHandler* definit a *src/net.cpp*. Aquest fil detectarà que al socket de la xarxa hi ha un missatge pendent de *A*. Llegirà el flux de bits i els emmagatzemarà en la variable *nRecvBytes*.

V. Periòdicament, *B* també executarà el fil *ThreadMessageHandler*. Aquest procés llegirà el flux de bits emmagatzemat en la variable *nRecvBytes*, comprovarà el tipus de missatge i executarà les funcionalitats corresponents a dit missatge (definides en la funció *ProcessMessage* del fitxer *src/main.cpp*).

```
else if (strCommand == "ping")
{
    if (pfrom->nVersion > BIP0031_VERSION)
    {
        uint64_t nonce = 0;
        vRecv >> nonce;
        // Echo the message back with the nonce. This allows for two useful features:
        //
        // 1) A remote node can quickly check if the connection is operational
        // 2) Remote nodes can measure the latency of the network thread. If this node
        //    is overloaded it won't respond to pings quickly and the remote node can
        //    avoid sending us more work, like chain download requests.
        //
        // The nonce stops the remote getting confused between different pings: without
        // it, if the remote node sends a ping once per second and this node takes 5
        // seconds to respond to each, the 5th ping the remote sends would appear to
        // return very quickly.
        pfrom->PushMessage("pong", nonce);
    }
}
```

Codi 39: resposta de Bitcoin Core a un missatge *ping*.

VI. Com a resposta al missatge *ping* d'*A*, *B* li tornarà un missatge *pong*. Per fer-ho, tal com es veu en el codi anterior, s'invocarà a la funció *PushMessage*, que guardarà a la variable *ssSend* (de la classe *Cnode* que representa A) el missatge *pong* a enviar.

VII. En la propera execució del fil *ThreadMessageHandler*, *B* llegirà la variable *ssSend*, codificarà el missatge en binari i l'emmagatzemarà per a la seva tramesa.

VIII. En la propera execució del fil *ThreadSocketHandler*, *B* enviarà, mitjançant el socket de la connexió amb *A*, el missatge codificat en binari.

Annex IV: llicències d'ús

Creative Commons Attribution 4.0 International Public License

By exercising the Licensed Rights (defined below), You accept and agree to be bound by the terms and conditions of this Creative Commons Attribution 4.0 International Public License ("Public License"). To the extent this Public License may be interpreted as a contract, You are granted the Licensed Rights in consideration of Your acceptance of these terms and conditions, and the Licensor grants You such rights in consideration of benefits the Licensor receives from making the Licensed Material available under these terms and conditions.

Section 1 – Definitions.

a. **Adapted Material** means material subject to Copyright and Similar Rights that is derived from or based upon the Licensed Material and in which the Licensed Material is translated, altered, arranged, transformed, or otherwise modified in a manner requiring permission under the Copyright and Similar Rights held by the Licensor. For purposes of this Public License, where the Licensed Material is a musical work, performance, or sound recording, Adapted Material is always produced where the Licensed Material is synched in timed relation with a moving image.
b. **Adapter's License** means the license You apply to Your Copyright and Similar Rights in Your contributions to Adapted Material in accordance with the terms and conditions of this Public License.
c. **Copyright and Similar Rights** means copyright and/or similar rights closely related to copyright including, without limitation, performance, broadcast, sound recording, and Sui Generis Database Rights, without regard to how the rights are labeled or categorized. For purposes of this Public License, the rights specified in Section 2(b)(1)-(2) are not Copyright and Similar Rights.
d. **Effective Technological Measures** means those measures that, in the absence of proper authority, may not be circumvented under laws fulfilling obligations under Article 11 of the WIPO Copyright Treaty adopted on December 20, 1996, and/or similar international agreements.
e. **Exceptions and Limitations** means fair use, fair dealing, and/or any other exception or limitation to Copyright and Similar Rights that applies to Your use of the Licensed Material.
f. **Licensed Material** means the artistic or literary work, database, or other material to which the Licensor applied this Public License.
g. **Licensed Rights** means the rights granted to You subject to the terms and conditions of this Public License, which are limited to all Copyright and Similar Rights that apply to Your use of the Licensed Material and that the Licensor has authority to license.
h. **Licensor** means the individual(s) or entity(ies) granting rights under this Public License.
i. **Share** means to provide material to the public by any means or process that requires permission under the Licensed Rights, such as reproduction, public display, public performance, distribution, dissemination, communication, or importation, and to make material available to the public including in ways that members of the public may access the material from a place and at a time individually chosen by them.
j. **Sui Generis Database Rights** means rights other than copyright resulting from Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, as amended and/or succeeded, as well as other essentially equivalent rights anywhere in the world.
k. **You** means the individual or entity exercising the Licensed Rights under this Public License. **Your** has a corresponding meaning.

Section 2 – Scope.

a. **License grant**.
    1. Subject to the terms and conditions of this Public License, the Licensor hereby grants You a worldwide, royalty-free, non-sublicensable, non-exclusive, irrevocable license to exercise the Licensed Rights in the Licensed Material to:
        A. reproduce and Share the Licensed Material, in whole or in part; and
        B. produce, reproduce, and Share Adapted Material.
    2. Exceptions and Limitations. For the avoidance of doubt, where Exceptions and Limitations apply to Your use, this Public License does not apply, and You do not need to comply with its terms and conditions.
    3. Term. The term of this Public License is specified in Section 6(a).
    4. Media and formats; technical modifications allowed. The Licensor authorizes You to exercise the Licensed Rights in all media and formats whether now known or hereafter created, and to make technical modifications necessary to do so. The Licensor waives and/or agrees not to assert any right or authority to forbid You from making technical modifications necessary to exercise the Licensed Rights, including technical modifications necessary to circumvent Effective Technological Measures. For purposes of this Public

License, simply making modifications authorized by this Section 2(a)(4) never produces Adapted Material.

5. Downstream recipients.
    A. Offer from the Licensor – Licensed Material. Every recipient of the Licensed Material automatically receives an offer from the Licensor to exercise the Licensed Rights under the terms and conditions of this Public License.
    B. No downstream restrictions. You may not offer or impose any additional or different terms or conditions on, or apply any Effective Technological Measures to, the Licensed Material if doing so restricts exercise of the Licensed Rights by any recipient of the Licensed Material.
6. No endorsement. Nothing in this Public License constitutes or may be construed as permission to assert or imply that You are, or that Your use of the Licensed Material is, connected with, or sponsored, endorsed, or granted official status by, the Licensor or others designated to receive attribution as provided in Section 3(a)(1)(A)(i).

b. **Other rights**.
    1. Moral rights, such as the right of integrity, are not licensed under this Public License, nor are publicity, privacy, and/or other similar personality rights; however, to the extent possible, the Licensor waives and/or agrees not to assert any such rights held by the Licensor to the limited extent necessary to allow You to exercise the Licensed Rights, but not otherwise.
    2. Patent and trademark rights are not licensed under this Public License.
    3. To the extent possible, the Licensor waives any right to collect royalties from You for the exercise of the Licensed Rights, whether directly or through a collecting society under any voluntary or waivable statutory or compulsory licensing scheme. In all other cases the Licensor expressly reserves any right to collect such royalties.

Section 3 – License Conditions.

Your exercise of the Licensed Rights is expressly made subject to the following conditions.

a. **Attribution**.
    1. If You Share the Licensed Material (including in modified form), You must:
        A. retain the following if it is supplied by the Licensor with the Licensed Material:
            i. identification of the creator(s) of the Licensed Material and any others designated to receive attribution, in any reasonable manner requested by the Licensor (including by pseudonym if designated);
            ii. a copyright notice;
            iii. a notice that refers to this Public License;
            iv. a notice that refers to the disclaimer of warranties;
            v. a URI or hyperlink to the Licensed Material to the extent reasonably practicable;
        B. indicate if You modified the Licensed Material and retain an indication of any previous modifications; and
        C. indicate the Licensed Material is licensed under this Public License, and include the text of, or the URI or hyperlink to, this Public License.
    2. You may satisfy the conditions in Section 3(a)(1) in any reasonable manner based on the medium, means, and context in which You Share the Licensed Material. For example, it may be reasonable to satisfy the conditions by providing a URI or hyperlink to a resource that includes the required information.
    3. If requested by the Licensor, You must remove any of the information required by Section 3(a)(1)(A) to the extent reasonably practicable.
    4. If You Share Adapted Material You produce, the Adapter's License You apply must not prevent recipients of the Adapted Material from complying with this Public License.

Section 4 – Sui Generis Database Rights.

Where the Licensed Rights include Sui Generis Database Rights that apply to Your use of the Licensed Material:

a. for the avoidance of doubt, Section 2(a)(1) grants You the right to extract, reuse, reproduce, and Share all or a substantial portion of the contents of the database;
b. if You include all or a substantial portion of the database contents in a database in which You have Sui Generis Database Rights, then the database in which You have Sui Generis Database Rights (but not its individual contents) is Adapted Material; and
c. You must comply with the conditions in Section 3(a) if You Share all or a substantial portion of the contents of the database.

For the avoidance of doubt, this Section 4 supplements and does not replace Your obligations under this Public License where the Licensed Rights include other Copyright and Similar Rights.

Section 5 – Disclaimer of Warranties and Limitation of Liability.

a. Unless otherwise separately undertaken by the Licensor, to the extent possible, the Licensor offers the Licensed Material as-is and as-available, and makes no representations or warranties of any kind concerning the Licensed Material, whether express, implied, statutory, or other. This includes, without limitation, warranties of title, merchantability, fitness for a particular purpose, non-infringement, absence of latent or other defects, accuracy, or the presence or absence of errors, whether or not known or discoverable. Where disclaimers of warranties are not allowed in full or in part, this disclaimer may not apply to You.

b. To the extent possible, in no event will the Licensor be liable to You on any legal theory (including, without limitation, negligence) or otherwise for any direct, special, indirect, incidental, consequential, punitive, exemplary, or other losses, costs, expenses, or damages arising out of this Public License or use of the Licensed Material, even if the Licensor has been advised of the possibility of such losses, costs, expenses, or damages. Where a limitation of liability is not allowed in full or in part, this limitation may not apply to You.

c. The disclaimer of warranties and limitation of liability provided above shall be interpreted in a manner that, to the extent possible, most closely approximates an absolute disclaimer and waiver of all liability.

Section 6 – Term and Termination.

a. This Public License applies for the term of the Copyright and Similar Rights licensed here. However, if You fail to comply with this Public License, then Your rights under this Public License terminate automatically.
b. Where Your right to use the Licensed Material has terminated under Section 6(a), it reinstates:
    1. automatically as of the date the violation is cured, provided it is cured within 30 days of Your discovery of the violation; or
    2. upon express reinstatement by the Licensor.

    For the avoidance of doubt, this Section 6(b) does not affect any right the Licensor may have to seek remedies for Your violations of this Public License.

c. For the avoidance of doubt, the Licensor may also offer the Licensed Material under separate terms or conditions or stop distributing the Licensed Material at any time; however, doing so will not terminate this Public License.
d. Sections 1, 5, 6, 7, and 8 survive termination of this Public License.

Section 7 – Other Terms and Conditions.

a. The Licensor shall not be bound by any additional or different terms or conditions communicated by You unless expressly agreed.
b. Any arrangements, understandings, or agreements regarding the Licensed Material not stated herein are separate from and independent of the terms and conditions of this Public License.

Section 8 – Interpretation.

a. For the avoidance of doubt, this Public License does not, and shall not be interpreted to, reduce, limit, restrict, or impose conditions on any use of the Licensed Material that could lawfully be made without permission under this Public License.
b. To the extent possible, if any provision of this Public License is deemed unenforceable, it shall be automatically reformed to the minimum extent necessary to make it enforceable. If the provision cannot be reformed, it shall be severed from this Public License without affecting the enforceability of the remaining terms and conditions.
c. No term or condition of this Public License will be waived and no failure to comply consented to unless expressly agreed to by the Licensor.
d. Nothing in this Public License constitutes or may be interpreted as a limitation upon, or waiver of, any privileges and immunities that apply to the Licensor or You, including from the legal processes of any jurisdiction or authority.