Analyse, valorisation et protection des réseaux pair-à-pair de blockchains publiques

Jean-Philippe Eisenbarth

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Analyse, valorisation et protection des réseaux pair-à-pair de blockchains publiques

THÈSE

présentée et soutenue publiquement le 13 décembre 2022

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Jean-Philippe Eisenbarth

Composition du jury

Président :	Vincent Chevrier	Professeur à l'ENSEM, université de Lorraine
Rapporteurs :	Arnaud Legout	Directeur de recherche Inria, centre Inria université Côte d'Azur
	Radu State	Professeur à l'université du Luxembourg
Examinateurs/rices :	Emmanuelle Anceaume	Directrice de recherche CNRS à l'IRISA, Rennes
	Maryline Laurent	Professeure à TELECOM SudParis, université Paris-Saclay
Encadrants :	Thibault Cholez	Maître de conférences à TELECOM Nancy, université de Lorraine
	Olivier Perrin	Professeur à l'IDMC, université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Je tiens tout d'abord à remercier les rapporteurs de cette thèse, Arnaud Legout et Radu State, pour le temps qu'ils ont consacré à mon travail. Je souhaite également remercier Emmanuelle Anceaume et Maryline Laurent pour l'intêret qu'elles ont porté à mes travaux en tant qu'examinatrices de cette thèse. Je remercie également Vincent Chevrier pour avoir présidé le jury et pour avoir été mon référent scientifique au Loria.

Je remercie très chaleureusement mes deux directeurs de thèse, Thibault Cholez et Olivier Perrin, pour m'avoir fait confiance depuis le début et m'avoir encadré durant ces trois années.

Je remercie également Ambroise Sander et Thomas Martignon, stagiaires dans l'équipe, pour leur contribution aux travaux de cette thèse et leur bonne humeur.

Je souhaite également remercier mes collègues dans l'équipe Resist et particulièrement mes collègues de bureaux Enzo, Philippe, Luke et Xavier.

Un grand merci à mes amis du Loria également, Quentin, Kevin, Sylvain, Omar, Matthieu, Victorien ainsi qu'à mes amis en dehors du laboratoire, les JJJ et les pionniers.

Je souhaite également remercier mes proches, mes parents, ma sœur, son compagnon, ma marraine ainsi que ma compagne et sa famille.

À toutes et à tous, merci !

Table des matières

Introduction Générale

1 Contexte

1.1 L'essor des blockchains

La technologie de la blockchain est apparue en 2008 avec la création de Bitcoin qui a pour principal objectif de fournir un service de paiement décentralisé autour d'une monnaie numérique. Plus précisément, une blockchain est un type de registre distribué géré par un réseau pair-à-pair (P2P) sous-jacent. Plus tard, Ethereum (2015) a proposé la notion de smart contract permettant l'exécution conjointe d'applications de manière décentralisée et convergeant vers un résultat validé par consensus. Depuis, ces blockchains ont beaucoup gagné en popularité et se retrouvent souvent au cœur de l'actualité ces dernières années, tant par leurs aspects informatiques qu'économiques. Par exemple, depuis février 2021, dans le canton de Zoug en Suisse, il est possible de payer ses impôts en bitcoin [1] ou encore en juin 2021, le Salvador a adopté le bitcoin comme monnaie ayant cours légal [2]. Ethereum a également connu un gain de popularité ces dernières années, notamment grâce aux NFTs (*Non-fungible tokens*, des jetons numériques non fongibles) et aux différents projets de Metaverse (monde numérique) qui les utilisent.

Beaucoup d'autres blockchains [3] sont apparues ultérieurement et, pour la plupart, elles essayent de modifier ou apporter une diversité sur certains aspects historiques des premiers systèmes. Par exemple, parmi ces différences figurent la réduction de la consommation énergétique du système, une véritable anonymisation des transactions, des valeurs de cours de cryptomonnaies stables et non volatiles ou encore un contrôle des utilisateurs (blockchains privées). Malgré cette popularité générale des blockchains et leur multiplication, Bitcoin et Ethereum restent de loin les plus importantes en termes de capitalisation totale (59% par rapport au top 100) et volume des transactions échangées en 24h (40%).

Comme nous l'avons énoncé plus tôt, les réseaux P2P sous-jacents à ces systèmes distribués servent à propager les informations à l'ensemble des nœuds les constituant. Plus précisément, les réseaux P2P sont un type de réseau informatique où chaque entité est une machine terminale qui est à la fois client et serveur. Cette architecture s'oppose de fait au modèle client-serveur comme illustré dans le schéma 1. Les principales forces de l'architecture P2P sont les faibles coûts d'infrastructure (car répartis entre tous les pairs), la forte tolérance aux pannes (par leur nature entièrement distribuée, sans point de défaillance unique) et un passage à l'échelle naturel (chaque pair contribuant au réseau à hauteur, en moyenne, de ce qu'il consomme). Les principales faiblesses découlent de biais potentiels, volontaires ou non, dans la constitution des pairs et

1. https://www.lagazettedescommunes.com/782227/pionniere-zoug-a-reussi-a-se-faire-un-nom-dans-les-cryptomonnaies

2. https://www.radiofrance.fr/franceculture/podcasts/le-pourquoi-du-comment-economie-et-social/pourquoi-le-salvador-a-t-il-decide-de-legaliser-le-bitcoin-5274533

3. On peut citer par exemple Tezos, Monero, Tether, Hyperledger.

surtout des difficultés de gestion et de contrôle de tels réseaux en l'absence d'autorité centrale pour encadrer les comportements déviants.

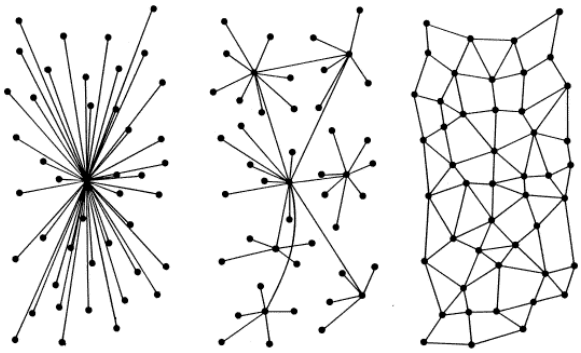

FIGURE 1 – Comparaison entre le modèle centralisé (client-serveur), décentralisé et distribué (P2P) (par Paul Baran, 1964).

1.2 La fiabilité relative des réseaux P2P

L'absence de gestion/contrôle centralisé ainsi que l'autonomie des pairs font que les réseaux P2P sont de fait vulnérables à diverses attaques (attaque Sybil, Eclipse, etc.). Ces attaques peuvent affecter durablement la fiabilité d'un réseau P2P et fournir un levier aux attaquants pour cibler l'application ou le service délivré. L'attaque Sybil, par exemple, consiste à insérer un grand nombre d'identités (fausse ou non) au sein du réseau pour déséquilibrer la part que représente une même entité (l'attaquant) dans le réseau. Cela peut permettre d'altérer le routage des messages, de polluer les données indexées, de surveiller les échanges (désanonymisation) ou encore de mener un déni de service. Historiquement, de telles attaques ont déjà été menées avec succès dans les années 2000 sur des réseaux P2P de partage de fichiers. Ces réseaux étaient pourtant bien plus vastes que ceux des blockchains actuelles dont ils partagent les mêmes structures. L'attaque Eclipse, quant à elle, est une évolution de l'attaque Sybil et consiste pour un attaquant à monopoliser l'ensemble des connexions d'un nœud du réseau avec ses propres nœuds. Ainsi, la victime est entièrement isolée du trafic du réseau P2P. Les applications d'une telle attaque sont multiples : double dépense de cryptomonnaie, déni de service, etc. Au-delà des attaques, les comportements égoïstes ou déviants des pairs ou certaines propriétés les concernant, comme leur distribution géographique ou leur fréquence de connexion/déconnexion, peuvent également constituer une menace pour la fiabilité des réseaux P2P. La détection et les mécanismes de protection contre de telles attaques ou comportement restent des défis majeurs encore aujourd'hui.

2 Problématique

L'essor de la technologie de blockchain a introduit plusieurs défis majeurs pour la communauté scientifique. Du point de vue de la sécurité de ces systèmes, nous pouvons séparer deux aspects : la fiabilité des mécanismes internes aux blockchains (structures, environnement d'exécution, algorithmes de consensus) et la robustesse et résilience des réseaux P2P sous-jacents et leur protocole. L'optimisation de ces systèmes s'ajoute à la sécurité comme autre défi majeur. Les réseaux P2P sont essentiels pour le bon fonctionnement des blockchains, mais ont une fiabilité toute relative. Or, ils demeurent très peu étudiés contrairement aux algorithmes de consensus.

Ainsi, la problématique de cette thèse peut être formulée de la manière suivante : **les réseaux P2P soutenant les principales blockchains publiques sont-ils fiables de par les propriétés des pairs qui les constituent et sont-ils utilisés à bon escient ? Sinon, quels outils ou correctifs proposer pour pallier les défauts constatés ?**

Le premier objectif de cette thèse est donc d'analyser les pairs constituant les réseaux P2P pour en déduire les propriétés. Pour cela, des architectures de supervision performantes doivent être utilisées. L'analyse des réseaux de Bitcoin et Ethereum revêt en effet plusieurs enjeux, à savoir la récupération rapide d'un maximum d'informations sur l'ensemble des pairs sans perturbation du fonctionnement du système. Les caractéristiques des pairs doivent permettre d'évaluer la fiabilité du réseau et le cas échéant proposer des mécanismes de protection. Le second objectif est d'améliorer les réseaux en optimisant l'utilisation qui en est faite tout en les protégeant des pairs suspects. La valorisation d'un réseau P2P dans le cadre d'une blockchain doit être transparente pour les utilisateurs et être compatible avec le fonctionnement actuel.

3 Organisation des contributions

Dans le cadre de cette thèse, **l'analyse des réseaux P2P des blockchains Bitcoin et Ethereum** est centrale et permet dans un premier temps d'identifier les caractéristiques, forces et faiblesses, de ces réseaux et dans un second temps **d'optimiser et sécuriser** leur utilisation. Ces deux contributions sont les parties principales de ce document, présentées dans les parties II et III. Un état de l'art est présenté en partie I. Nous terminons ce document par une conclusion générale accompagnée de quelques perspectives de recherches.

Première partie : État de l'art

Dans le chapitre 1, nous présentons le fonctionnement général des systèmes de blockchain, et en particulier Bitcoin et Ethereum. Nous motivons également notre choix d'étudier les caractéristiques des réseaux P2P sous-jacents à ces deux systèmes.

Dans le chapitre 2, nous nous intéressons à la sécurité de Bitcoin et d'Ethereum, tant du point de vue des communications que des mécanismes de consensus. Plus particulièrement, l'attaque Sybil est un des sujets principaux d'étude et nous constatons le manque de mécanismes de protection satisfaisants.

Deuxième partie : Supervision et analyse des réseaux P2P de Bitcoin et Ethereum

Le chapitre 3 présente de manière détaillée le fonctionnement du réseau P2P non structuré de Bitcoin et la mise en œuvre de notre architecture de supervision. Les données récoltées dans notre campagne de mesure sont ensuite analysées et les caractéristiques que nous avons dérivées de celles-ci sont présentées. Ces caractéristiques nous permettent de fournir une appréciation de la fiabilité du réseau.

Dans le chapitre 4, nous étudions le fonctionnement du réseau P2P d'Ethereum et plus particulièrement l'implantation de sa DHT. Nous présentons également notre architecture de supervision qui tire parti de la structure du réseau induite par la DHT et permet la découverte rapide des pairs sans nuire au bon fonctionnement du réseau. Cette supervision nous permet d'analyser les caractéristiques, forces et faiblesses du réseau à travers l'analyse des données récoltées.

Troisième partie : Stockage efficace des données et prévention d'attaques Sybil sur Ethereum

L'analyse de la DHT d'Ethereum nous permet de constater sa sous-utilisation et de travailler, dans le chapitre 5, à sa valorisation. Nous proposons une nouvelle architecture de stockage optimisée des données d'Ethereum tirant parti de la DHT et permettant de réduire significativement les données stockées par chaque pair.

Par de nouvelles analyses spécifiques, nous mettons en exergue, dans le chapitre 6, des comportements suspects de type Sybil de certains pairs du réseau P2P d'Ethereum, prouvant ainsi la vulnérabilité du réseau à ce type d'attaque.

Pour finir, dans le chapitre 7, nous proposons un système de prévention des attaques Sybils sur Ethereum. Celui-ci est basé sur notre outil de supervision, couplé à un système de propagation des informations des pairs suspects et un mécanisme de révocation complètement distribué.

4 Terminologie

Dans la suite de ce manuscrit, nous utiliserons un ensemble de termes dont nous donnons une définition ici :

- Un **utilisateur** est un individu (ou une entité) utilisant un client P2P ;
- Un **client P2P** est un logiciel qui fournit un service de connexion et d'interactions dans une architecture P2P ;
- Les termes **nœud** et **pair** désigne une machine qui exécute une instance d'un client P2P, une telle machine étant identifiée par un couple adresse IP–port ou un identifiant P2P ;
- Un **protocole P2P**, dont un nom plus long est protocole de communication P2P, désigne la spécification des règles de communications que les machines doivent suivre au sein d'un même réseau P2P ;
- Un **réseau P2P** désigne l'ensemble des pairs communiquant par l'utilisation d'un même protocole et qui sont interconnectés ;
- On appelle **voisins** les pairs auxquels un nœud est directement connecté au sein du réseau.
- Un **système de blockchain** désigne l'association des structures de données, des algorithmes de consensus, et du réseau P2P pour réaliser un registre distribué.

Il existe plusieurs états dans lequel un nœud peut se trouver. Il peut être soit joignable, ce qui signifie qu'il est public et routé sur Internet, soit inaccessible. Un nœud inaccessible peut être simplement hors ligne ou être en ligne, mais non accessible publiquement (derrière un NAT (Network Address Translation) ou un pare-feu, par exemple). Dans ce dernier cas, il peut toujours participer au réseau par le biais de connexions sortantes, mais il n'accepte pas les connexions entrantes et seuls les nœuds avec lesquels il a établi une connexion savent qu'il est en ligne. Dans le cadre des blockchains, un nœud peut exécuter soit un client logiciel complet, soit un client léger. Dans le premier cas, le nœud va stocker l'ensemble des données de la blockchain et contribuer au réseau en les validant et en les propageant au sein du réseau. Dans le second cas, le nœud ne stockera que les entêtes des données et devra demander à un nœud complet les informations complémentaires pour permettre à un utilisateur d'émettre une transaction.

Des termes applicables uniquement à un système de blockchain en particulier seront définis en début des chapitres qui en feront l'usage.

Première partie

État de l'art

Chapitre 1

Bitcoin et Ethereum, deux blockchains publiques

Sommaire

Introduction

En 2022, Bitcoin et Ethereum sont les deux blockchains les plus populaires, en termes de capitalisation totale et volume de cryptomonnaie en circulation [4], Bitcoin étant présenté comme un système de paiement électronique décentralisé, Ethereum mettant plutôt son système de « contrats intelligents » en avant. Ces deux systèmes sont construits par la combinaison de structures de données complexes, d'algorithmes et mécanismes de consensus, et d'architectures de systèmes distribués. Tous ces éléments sont critiques pour le bon fonctionnement de ces systèmes.

Dans la suite de ce chapitre, nous allons présenter la structure et les mécanismes des blockchains de Bitcoin et Ethereum, ainsi que les architectures des réseaux pair-à-pair sous-jacents à celles-ci. Pour finir, nous présenterons les points sur lesquels Bitcoin et Ethereum se démarquent ou se rejoignent et ce que cela implique en termes de robustesse.

4. `https://coinmarketcap.com/`

1.1 Principes généraux de fonctionnement d'une blockchain

Les grands principes des blockchains que nous connaissons aujourd'hui ont été décrits en 2008 par Nakamoto [5] dans son livre blanc sur un système de paiement électronique pair-à-pair appelé Bitcoin [Nak08], puis ces principes ont été affinés et les fonctionnalités de ces systèmes ont été étendues par Gavin Wood et Vitalik Buterin [Woo14] [But14] pour créer Ethereum. Ces systèmes de paiement électronique fonctionnent comme des registres distribués qui gèrent une liste d'enregistrements (transactions) protégés contre la falsification ou la modification. Les actifs numériques échangés s'appellent des crypto-monnaies.

Après les publications de Nakamoto, de Wood et Buterin, les chercheurs du monde entier se sont intéressés à l'étude des blockchains sous divers aspects, comme la robustesse ou l'utilisabilité, et notamment Gramoli et al. [Gra20] propose une synthèse des problèmes de consensus sur les blockchains ainsi qu'une formalisation des règles de consensus de Bitcoin et Ethereum.

1.1.1 Qu'est-ce qu'une blockchain ?

Une blockchain est constituée par un ensemble de blocs reliés entre eux. Un bloc donné n'est lié qu'à son bloc parent (bloc précédent). Le bloc enfant contient un pointeur (en pratique le hash du contenu du bloc parent) qui le relie à un bloc parent. Par exemple un bloc B1 (enfant) contient le hash du contenu du bloc B0 (parent), on dit que B1 pointe vers B0. La blockchain grandit alors à mesure que de nouveaux blocs y sont ajoutés. Le tout premier bloc sert de point de départ pour la blockchain et est appelé *genesis block* (bloc de genèse). Dans sa forme la plus simple, une blockchain est une liste chaînée.

Chaque bloc de la blockchain contient des transactions, qui essentiellement contiennent les informations d'un émetteur et récepteur ainsi qu'un montant en cryptomonnaie. En prenant en compte l'ensemble des transactions de la blockchain, nous avons un registre qui définit la balance de chaque compte du système. Chaque utilisateur du système possède un compte (on parle aussi d'adresse) et peut effectuer des transactions vers d'autres utilisateurs. Pour sécuriser les transactions, on utilise la cryptographie asymétrique : chaque utilisateur génère une paire de clés, une clé publique et une privée, et signe ses transactions avec sa clé privée, le destinataire pouvant alors vérifier l'expéditeur en vérifiant la signature avec la clé publique de l'expéditeur présumé. Les utilisateurs utilisent un portefeuille de crypto-monnaie (*cryptocurrency wallet* en anglais) pour gérer cela. Un portefeuille de crypto-monnaie est un dispositif, physique ou logiciel, qui stocke les clés publiques et privées de l'utilisateur et peut être utilisé pour suivre la propriété, recevoir ou dépenser des crypto-monnaies. Une transaction dont la validité a été vérifiée [6] doit ensuite être encapsulée au sein d'un bloc pour être confirmée. En effet, les utilisateurs doivent pouvoir se mettre d'accord sur un ordre total des transactions pour garder toujours le même historique des dépenses. Les mineurs sont responsables d'encapsuler les transactions dans les blocs qu'ils génèrent. Lorsqu'un nouveau bloc est créé (un certain nombre de transactions y figurent) il doit ensuite être envoyé à tous les utilisateurs pour qu'ils mettent à jour leur balance locale afin que l'ensemble du réseau partage le même état global. Ce système est géré de manière distribuée, c'est-à-dire que les utilisateurs sont également acteurs et forment un réseau pair-à-pair pour transmettre les nouvelles transactions et nouveaux blocs. Lorsque le réseau P2P est public, alors le système est accessible à tout le monde. On oppose à cela les blockchains privées qui nécessitent généralement l'intervention d'un tiers de confiance pour déclarer et approuver les

5. Son nom est un pseudonyme et en 2022 nous ne connaissons toujours pas sa véritable identité.

6. L'émetteur possède bien la clé privée associée à son compte et il possède bien les cryptomonnaies qu'il souhaite transférer.

participants du réseau.

Un des principaux problèmes qui a dû être résolu avec les blockchains publiques est ce qu'on appelle un fork. Un fork arrive lorsque deux mineurs différents propagent chacun, dans un intervalle de temps proche, un bloc dont le parent est identique. La blockchain cesse alors d'être une liste chaînée et devient un arbre ou plus précisément un graphe orienté acyclique (*Directed Acyclic Graph, DAG* en anglais). Un exemple de représentation d'une blockchain sous la forme d'un DAG est illustré dans le schéma 1.1. On peut y observer le bloc de genèse, qui est le bloc initial de la blockchain, et deux forks, au niveau 2 et 3 à partir de la gauche, où plusieurs blocs sont présents au même niveau du DAG, signifiant qu'ils partagent le même bloc parent. Il y a également d'autres éléments (blocs confirmés, incertains et orphelins) sur ce schéma que nous expliciterons dans la section suivante.

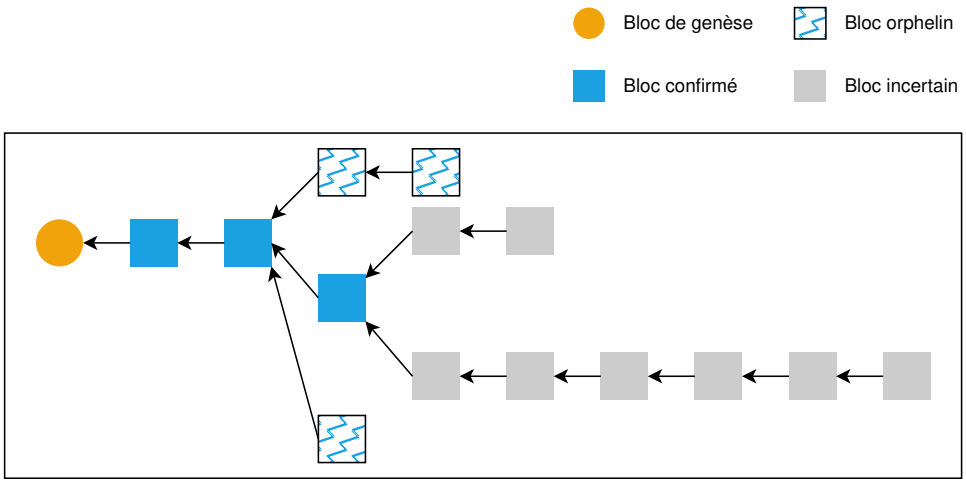

FIGURE 1.1 – Représentation schématique d'une blockchain. Ici, la règle de sélection de la chaîne est celle utilisée par les mécanismes de consensus de Bitcoin.

L'ensemble des utilisateurs doit alors se mettre d'accord pour définir quels blocs prendre vraiment en compte (résoudre le fork). Les règles misent en œuvre pour cela sont appelés des mécanismes de consensus.

1.1.2 Mécanismes de consensus

Dans le cadre des blockchains publiques, tout le monde peut participer au minage des blocs et pour la bonne santé du système, il faut pouvoir éviter qu'une même entité monopolise toutes les ressources et soit la seule à pouvoir encapsuler les transactions dans les blocs, car elle aurait alors le contrôle total du système. Pour résoudre ce problème, le ou les créateurs de Bitcoin ont implanté un système dont les bases avaient déjà été décrites en 1992 par Dwork et Moni [DN93] et étendues en 1999 par Jakobsson et Juels [JJ99] (donnant également le nom de *proof-of-work* à cette technique). L'idée était d'empêcher les attaques de type déni de service par une entité monopolisant les ressources du réseau. Ainsi, un mineur doit résoudre un crypto-puzzle, à base de calcul de hash jusqu'à un certain objectif de difficulté définit en pratique par l'obtention d'un hash avec un nombre de bits de poids fort consécutifs, et s'il y arrive, il a alors la possibilité de produire un nouveau bloc et de remporter une récompense pour le travail fourni. Il inclut dans son bloc une preuve (d'où le nom « preuve de travail ») qu'il est bien l'auteur de la résolution du crypto-puzzle (à l'aide de sa clé privée) et l'ensemble du réseau peut vérifier la conformité du

bloc produit. Voici un exemple simplifié d'une preuve de travail :

```
Difficulté = 5 bits

hash(bloc, nonce1) = 000011... - Preuve non valide
hash(bloc, nonce2) = 0000011... - Preuve valide
```

La difficulté s'ajuste automatiquement afin de garantir un temps constant entre deux blocs consécutifs. Par exemple, pour Bitcoin, il a été défini que le temps entre deux nouveaux blocs doit être d'environ 10 minutes et la difficulté s'ajuste tous les 2016 blocs (2 semaines environ) pour le garantir. Si les 2016 blocs ont été minés plus rapidement que prévu, la difficulté augmente et inversement. Du fait de la forte difficulté de résolution des crypto-puzzle pour un mineur isolé et du nombre croissant de mineurs dans le réseau (et donc d'une plus forte concurrence), ils se sont rassemblés au sein de coopératives de mineurs (*mining pool*) où ils mutualisent les coûts et partagent les récompenses. En théorie, une entité ou une coalition d'entités qui arriveraient à détenir plus de 51% de la puissance de calcul du réseau pourrait prendre le contrôle de la blockchain, mais dans la pratique, rassembler plus de 51% de la puissance de calcul pour un seul mineur est très couteux [JJ99] [Nak08], sauf maladresse d'une coopérative de minage. En effet, en 2014, la coopérative de mineurs « GHash.io » (pour Bitcoin) s'est dangereusement approchée d'une puissance de 51% mais la communauté a été alerté par quelques acteurs qui avaient déployé des techniques de supervision et rapidement les mineurs ont quitté cette coopérative. Les travaux de Sayeed et al. [SM19] montre qu'une coalition de mineur est une menace très sérieuse pour les blockchains basées sur la preuve de travail. Ethereum a aussi implanté la preuve de travail pour le minage des blocs, mais depuis fin 2020 la communauté a commencé à faire une transition vers une nouvelle technique appelée *proof-of-stake* (deux traductions sont possibles en français : preuve d'enjeu ou preuve de participation). Le 15 septembre 2022, cette transition a été achevée par l'événement *The Merge*, la fusion de la *Beacon Chain* (la chaîne *proof-of-stake* débutée en 2020) avec la chaîne principale *proof-of-work*. La preuve d'enjeu consiste à former un ensemble de validateurs à partir des utilisateurs qui ont mis en dépôt 32 Ether (la cryptomonnaie d'Ethereum) et de choisir aléatoirement au sein de cet ensemble un validateur qui sera responsable de produire le prochain bloc. Cette technique est plus récente que la preuve de travail et donc sa robustesse n'a pas été autant étudiée et dépasse le cadre de cette thèse.

La preuve de travail n'est pas suffisante pour garantir le bon fonctionnement de la blockchain, notamment face à l'incertitude de la branche du DAG à suivre lorsqu'il y a un fork (voir schéma 1.1). En effet, un fork peut être la cause d'une attaque de double dépense : un attaquant peut dépenser deux fois les mêmes crypto-monnaies si les transactions correspondantes se trouvent dans des blocs situés dans deux embranchements différents du DAG. Pour résoudre cela, il faut utiliser une règle permettant de choisir de façon déterministe la liste chainée (la blockchain en tant que tel, aussi appelée la chaîne principale) à prendre en compte pour appliquer les prochaines transactions. Bitcoin possède un des mécanismes de consensus le plus simple dit *longest chain rule* : la chaîne principale est celle formée par le plus grand nombre de blocs reliés entre eux. Cette chaîne principale est celle qui contient alors les blocs confirmés, c'est-à-dire les blocs qui contiennent les transactions à prendre en compte définitivement. Les autres blocs, appelés orphelins, sont à ignorer dans le cas de Bitcoin. Ces mécanismes de consensus sont parfois appelés « Nakamoto Consensus ». En complément de cette règle de la chaîne la plus longue, un système de paiement en Bitcoin définit un nombre de blocs à attendre avant d'entériner les transactions de la chaîne principale. Plus la transaction est importante, plus il est judicieux d'attendre un grand nombre de blocs avant de la considérer enregistrée, car à cause

des forks, il est tout à fait possible qu'à un instant donné une branche du DAG l'emporte, mais quelques blocs plus tard, c'est une autre branche qui la supplante. Lorsqu'il s'agit de petites transactions, les services de paiement autorisent ce que l'on appelle du *zero-confirmation*, c'est-à-dire que la transaction est entérinée une fois qu'elle est incluse dans un bloc qui fait partie de la chaîne principale (environ 10 minutes d'attente). Pour les transactions plus grandes, un service de paiement définira généralement 6 blocs de confirmation (environ 1h d'attente), afin d'être sûr que les transactions peuvent bien être entérinées. Pour reprendre l'exemple du schéma 1.1, on peut considérer que l'on peut entériner les transactions des trois premiers blocs confirmés (bleus), car ils appartiennent bien à la chaîne principale et ils ont bien au minimum 6 blocs descendants. Si le prochain bloc à être miné est ajouté à la suite de la chaîne des 6 blocs gris alors le premier de cette chaîne serait confirmé (et on pourrait lui appliquer la couleur bleue). Ethereum possède des mécanismes plus complexes, que nous détaillons dans la section 1.3.

La question de la diffusion des blocs et des transactions à l'ensemble du réseau P2P se pose alors et pour ce faire, Bitcoin et Ethereum ont choisi des architectures différentes. Dans la suite, nous allons nous intéresser à celles-ci et donner leurs avantages et inconvénients.

1.2 Architectures des réseaux pair-à-pair sous-jacents

Parmi les architectures de réseaux P2P, on peut distinguer deux familles : les réseaux P2P avec et sans serveur. Dans la plupart des implantations des réseaux P2P des blockchains publiques, et particulièrement celles de Bitcoin et Ethereum, l'architecture P2P sans serveur est choisie. Cette architecture permet de former des réseaux P2P complètement distribués sans autorité centrale qui introduirait des problèmes de confiance dans le système. Bitcoin et Ethereum se distinguent tout de même, car au sein des architectures P2P sans serveur, il existe les réseaux P2P non structurés, tel que Bitcoin, et les réseaux P2P structurés, comme Ethereum.

1.2.1 Réseau pair-à-pair non structuré de Bitcoin

Un réseau P2P non structuré se caractérise par l'absence d'une quelconque organisation des pairs : ils établissent aléatoirement des liens entre eux. Chaque pair propage alors les messages qu'il reçoit à tous les voisins auxquels il est connecté. On appelle cela une propagation par inondation (*flooding* en anglais). Le schéma 1.2 illustre ce fonctionnement. Le réseau P2P de partage de fichiers Gnutella, créé en l'an 2000 [Thei], fut un des premiers largement déployés basé sur cette architecture.

La propagation par inondation passe cependant très mal à l'échelle, car le nombre de messages émis augmente linéairement $\mathcal{O}(N)$ avec N, le nombre de pairs dans le réseau. Ce fut un problème pour Gnutella qui a introduit divers mécanismes (marche aléatoire, super-pairs) pour réduire le nombre de messages émis pour une recherche de fichier. Ce problème ne se pose pas pour Bitcoin dont la finalité du réseau est précisément de contacter tous les pairs pour la synchronisation. Les développeurs de Bitcoin ont tout de même optimisé légèrement le protocole en implantant des messages d'annonces de nouveaux blocs ou transactions pour que les pairs ne les propagent seulement aux pairs qui ne les ont pas déjà. En effet, il existe quatre principaux types de messages dans le réseau P2P de Bitcoin :

— les messages de découverte du réseau ;
— les annonces de transactions ou de blocs ;
— les demandes de transactions ou de blocs ;
— l'envoi des transactions ou blocs.

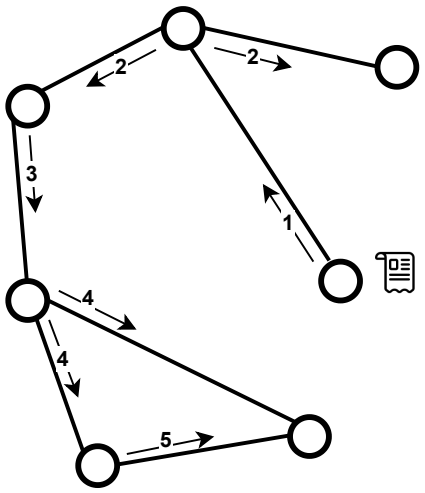

FIGURE 1.2 – Illustration du mécanisme de transmission de messages par inondation (*flooding protocol*).

Les messages de découverte du réseau sont détaillés dans le chapitre 3 dans lequel on présente une analyse du réseau P2P. Les annonces et demandes de transactions/blocs permettent d'optimiser le nombre et la taille des messages envoyés sur le réseau. Lorsqu'un nœud reçoit une nouvelle ressource, il l'annonce à ses voisins, lesquels doivent explicitement lui demander qu'il la lui transmette. Le cas échéant, la transaction ou le bloc (qui est une ressource relativement volumineuse) est effectivement envoyé. Les auteurs de [DW13] se sont intéressés au temps de propagation d'un bloc sur le réseau. Le temps médian est de 6,5 secondes tandis que le temps moyen est de 12,6 secondes et 5% des nœuds du réseau P2P reçoivent les blocs plus de 40 secondes après leur émission. Ces temps de propagation assez longs facilitent des attaques tirant parti de forks dans la blockchain.

1.2.2 Réseau pair-à-pair structuré d'Ethereum

À la différence d'un réseau P2P non structuré, un réseau P2P structuré présente une organisation efficace du réseau. En effet, chaque pair du réseau dispose d'un identifiant [7] permettant de le retrouver, en un minimum de messages, en suivant un chemin déterministe parmi les pairs. Cet identifiant n'est pas choisi arbitrairement, il est généralement le résultat d'une fonction de hachage. Une ressource partagée par le réseau peut également posséder un identifiant, permettant de localiser rapidement le ou les pairs responsables du stockage de cette ressource. La structure de données qui représente cela s'appelle une table de hachage distribuée (en anglais *Distributed Hash Table*, DHT).

La force des DHT réside dans la localisation efficace des pairs et des ressources du réseau. Pour trouver un pair ou une ressource, $\mathcal{O}(\log_2 N)$ messages au maximum sont nécessaires, N étant le nombre de pairs au sein du réseau. En comparaison des réseaux P2P non structurés, dont le nombre de messages pour localiser un pair augmente linéairement $\mathcal{O}(N)$, le routage de proche en proche des réseaux P2P structurés est bien plus performant. Une autre propriété des DHT très intéressante (du point de vue de la résilience) pour un réseau P2P est la répartition de la charge de manière totalement homogène sur l'ensemble des pairs.

7. Aussi appelé « ID réseau » dans la suite de ce manuscrit.

Les développeurs d'Ethereum ont fait le choix d'utiliser Kademlia comme DHT pour le réseau P2P. Nous verrons dans le chapitre 4 comment fonctionne en détail le réseau P2P d'Ethereum et en particulier les différences d'implantation avec Kademlia. Ici, nous allons décrire le fonctionnement théorique de Kademlia.

Kademlia [MM02] a été décrit en 2002 et a été largement utilisé dans l'implantation de DHT de divers réseaux P2P, certains encore utilisés actuellement en 2022. Historiquement, on peut citer KAD (eMule) et Mainline DHT (Bittorrent) comme implantations populaires ayant regroupé plusieurs millions d'utilisateurs simultanés. Plus récemment, on peut citer Ethereum (2015) et IPFS (2015). Les identifiants des pairs et ressources sont répartis sur un espace d'adressage à 160 bits et pour les localiser, une métrique basée sur l'opérateur logique XOR (OU exclusif) permettant d'évaluer la distance entre deux identifiants est proposée. Le schéma 1.3 montre l'utilisation de cette métrique avec le calcul de quatre distances entre le pair 5 (0111) et quatre autres pairs sur un espace d'adressage à 4 bits. L'opérateur XOR possède trois propriétés qui le rendent très intéressant comme métrique de mesure de distance :

— A XOR A = 0, la distance d'un pair à lui-même est zéro ;
— A XOR B = B XOR A, l'opération est commutative (ou symétrique) ;
— A XOR B ≤ (A XOR C) + (C XOR B), l'opération respecte l'inégalité triangulaire.

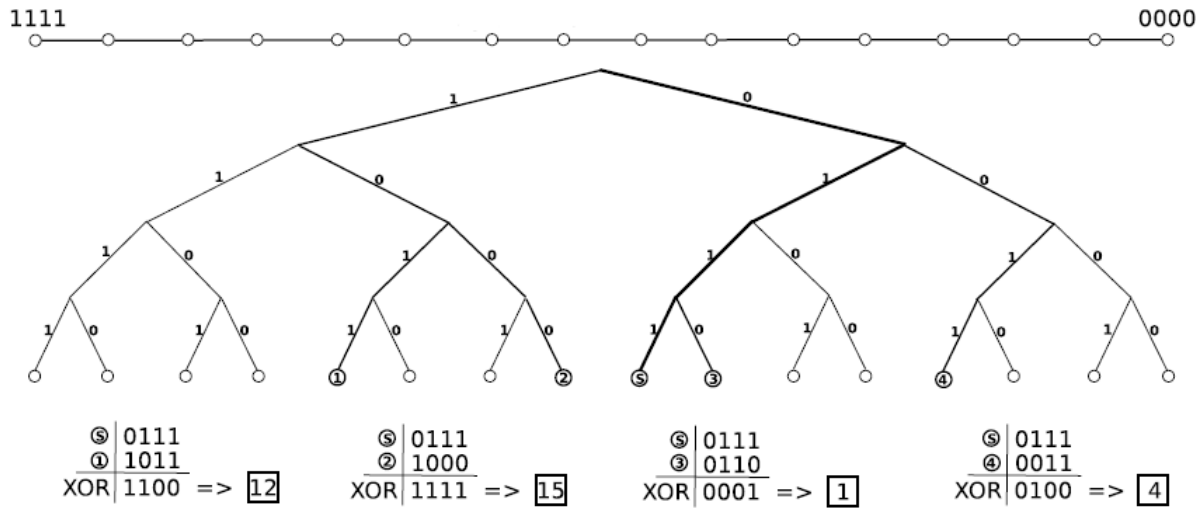

FIGURE 1.3 – Calcul de la distance entre le pair 5 et d'autres pairs dans un espace d'adressage à 4 bits à l'aide de la métrique XOR, d'après René Brunner [BB06]

Le protocole de Kademlia est composé de 4 types de messages, à savoir `PING`, `STORE`, `FIND_NODE`, `FIND_VALUE`. Le premier permet simplement de tester si un pair est en ligne et accessible, les trois autres types de messages concernent directement la DHT. Étant donnée k, la constante de réplication du système (par exemple $k = 10$), le message `STORE` permet de stocker une donnée sur les k pairs les plus proches de l'identifiant de la donnée en utilisant la métrique XOR. Ces k pairs retourneront cette donnée lors d'une recherche sur l'identifiant à l'aide du message `FIND_VALUE`. `FIND_NODE` est un message de découverte de nœuds, retournant les k pairs les plus proches de l'identifiant passé en paramètre. Pour le bon fonctionnement de ce protocole, chaque pair construit sa propre table de routage qui peut être représentée comme un arbre binaire dont chaque niveau i contient au plus k contacts (cette structure est appelée *k-bucket*). Pour chaque niveau $0 \leq i < 160$, le pair courant stocke les contacts situés à une distance comprise entre 2^i et 2^{i+1}. Le schéma 1.4 illustre une telle structure. Un exemple de recherche itérative d'un pair,

aussi appelé *lookup* en anglais, est illustré dans le schéma 1.5.

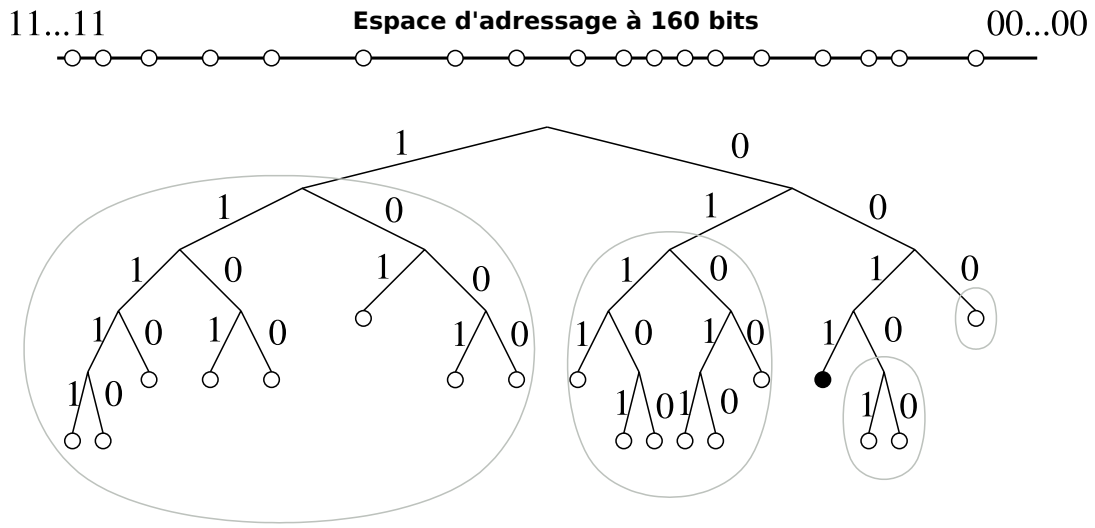

FIGURE 1.4 – Extrait de la table de routage du pair 0011 sous la forme d'un arbre binaire, les ovales grises montrent les k-buckets dans lesquels le pair courant (0011) doit connaitre au moins un contact, d'après Maymounkov et al. [MM02].

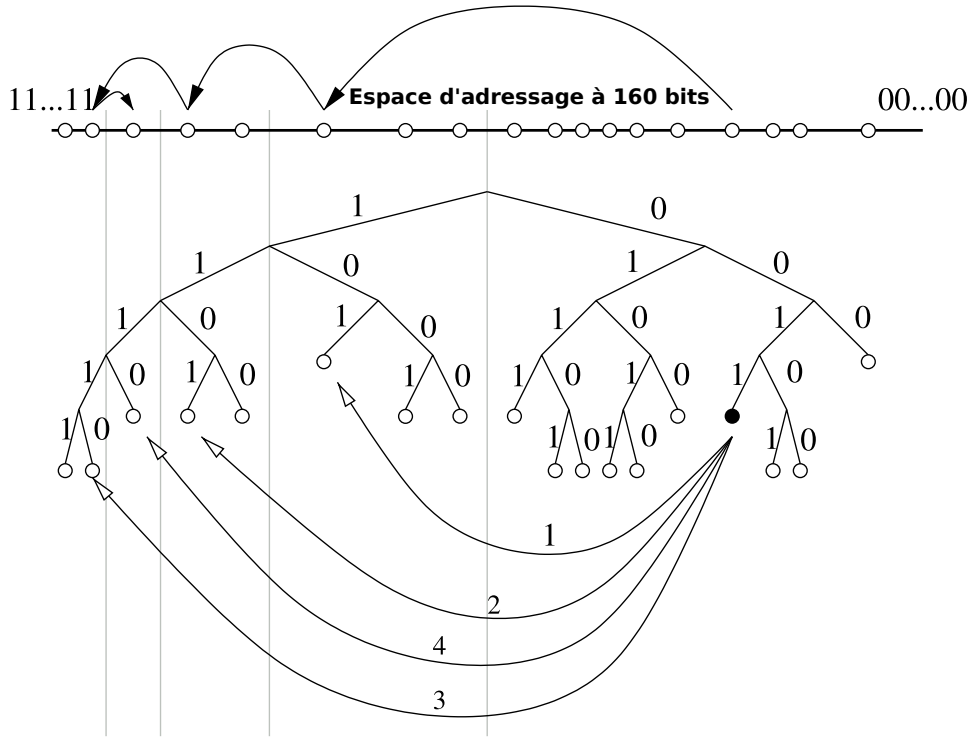

FIGURE 1.5 – Schéma de la procédure de *lookup* itératif pour trouver un nœud à partir de son identifiant, d'après Maymounkov et al. [MM02].

Le nœud, dont le préfixe est 0011, cherche à trouver le nœud de préfixe 1110 en effectuant des requêtes itératives vers les nœuds de plus en plus proches, la structure même des tables

de routage augmentant naturellement la précision à mesure qu'on se rapproche de la cible. La première requête est effectuée vers le nœud 101, qui donne les informations des contacts les plus proches de la cible dont il a connaissance. La seconde requête est alors effectuée vers 1101 qui, de même, va donner les informations du nœud 11110 qui lui-même donnera les informations de la cible 1110, car elle est un de ses contacts connus. Une requête `FIND_VALUE` pourrait dès lors être utilisée pour récupérer une donnée stockée par 1110.

La DHT d'Ethereum, dérivée de Kademlia, est implantée au sein d'un protocole nommé *Node Discovery*, et nous détaillons son fonctionnement dans le chapitre 4.

1.3 Similarités et différences entre Bitcoin et Ethereum

1.3.1 Crypto-monnaies en circulation et récompenses

Même si Bitcoin et Ethereum utilisent la preuve de travail comme mécanisme de consensus, tous deux ont choisi des stratégies différentes pour récompenser les mineurs. Au départ, un mineur Bitcoin recevait 50 BTC (+ frais des transactions), créés *ex-nihilo*, s'il parvenait à être le plus rapide à résoudre le crypto-puzzle et à imposer son bloc au réseau. Mais Nakamoto a prévu un nombre fixe de bitcoins en circulation, 21 000 000, et pour imposer cela, la récompense offerte pour la création d'un nouveau bloc est divisé par 2 tous les quatre ans. On appelle cela le *halving*. En 2022, cette récompense est de 6,25 BTC et il y a un peu plus de 19 000 000 BTC en circulation [8]. Les créateurs d'Ethereum, eux, n'ont pas prévu de limite d'ethers en circulation, il n'y a pas de *halving*. La récompense offerte aux mineurs est de 2 ETH (+ frais des transactions) et il y a un peu plus de 122 500 000 ETH en circulation[8].

1.3.2 *Epoch*

Les deux blockchains définissent une période de temps caractéristique, appelée *epoch*, mais elle n'a pas du tout la même signification. Pour Bitcoin, une *epoch* correspond à la période entre deux *halving*, elle équivaut donc à 4 ans. Pour Ethereum, l'*epoch* actuelle correspond aux 30 000 derniers blocs de la blockchain. Elle est utilisée comme unité dans le client de référence Geth pour définir à quel moment les blocs Ethereum, stockés dans une base de données rapide (LevelDB), sont transférés dans une base de données long terme (FreezeDB). Il s'agit des blocs des trois dernières *epoch* qui sont conservés dans LevelDB et ceux au-delà sont transférés et conservés dans FreezeDB. Nous approfondissons le fonctionnement du stockage d'Ethereum dans le chapitre 5, dans lequel nous proposons une nouvelle solution de stockage des blocs de la base de données de long terme.

1.3.3 *Smart contracts*

Lorsque l'on oppose Bitcoin et Ethereum, il s'agit souvent de différences entre les services proposés. Bitcoin a été la première crypto-monnaie créée et est utilisée en tant que telle, c'est-à-dire comme un système de paiement décentralisé. Ethereum, quant à elle, a apporté le support de *smart contracts* (« contrats intelligents ») que l'on peut développer avec un langage de programmation Turing-complet appelé Solidity. Ces contrats intelligents sont exécutés de manière totalement distribuée par le réseau. Les applications sont très variées : systèmes de traçabilité, systèmes de sécurité (contrôle d'accès par exemple), jeux, organisations autonomes décentralisées (DAO), etc. Bitcoin possède également un langage, appelé Script, permettant d'écrire des smart

8. `https://coinmarketcap.com/`

contracts mais ce langage n'est pas Turing-complet et ne permet donc pas la même diversité d'applications qu'Ethereum.

Plus précisément, un smart contract d'Ethereum est un programme, avec un ensemble de fonctions et d'états, réuni au sein d'un compte Ethereum (nous détaillons les notions d'états et compte dans le chapitre 5). Ce compte possède une adresse publique, une balance et peut recevoir des transactions. Un smart contract n'est pas contrôlé par un utilisateur, ils sont déployés et exécutés automatiquement par les nœuds du réseau lorsqu'un utilisateur lui envoie une transaction pour exécuter une de ses fonctions. Une unité particulière est utilisée au sein d'Ethereum pour mesurer le travail effectué (exécution d'une fonction d'un smart-contract par exemple), il s'agit du gaz (*gas* en anglais). Le prix du gaz est exprimé en gwei (une subdivision de l'ether, égale à 10^{-9} ETH) et il existe une limite de gaz dépensable en une transaction (21000) et une limite par bloc (jusqu'à 30 millions). Ce prix en gaz est payé par l'utilisateur qui émet une transaction et il est récupéré en récompense par le mineur qui a remporté la compétition pour le minage du bloc courant (il remporte l'ensemble du gaz des transactions qu'il a choisi d'inclure dans le bloc miné)

1.3.4 Règle de sélection de la chaîne principale (mécanisme de consensus)

Bitcoin implante la règle de sélection dite de « la chaîne la plus longue », c'est-à-dire que la chaîne principale est celle formée par le plus grand nombre de blocs reliés entre eux. Sur le schéma 1.6, si ce DAG représentait la blockchain Bitcoin, la chaîne principale (les blocs dits « confirmés ») serait $b1 \leftarrow b3 \leftarrow b7 \leftarrow b9 \leftarrow b11$ et tous les autres blocs seraient orphelins, et les transactions qu'ils contiendraient ne seraient pas prises en compte. Avec cette règle, la puissance de calcul des mineurs qui ont produit les blocs qui finissent finalement orphelins (à partir de b2 sur le schéma) est gâchée et n'apporte rien à la blockchain. De plus, les auteurs de [SZ15] ont montré qu'en présence de nombreux forks, cette règle était vulnérable à des attaques de type *selfish mining* (voir chapitre 2). Ils proposent un nouveau protocole, appelé GHOST (*Greedy Heaviest-Observed Sub-Tree*), pour répondre à ce problème. D'autres études [KP19] [Gra20] ont montré sa validité et prouvé sa sécurité. Visuellement, l'idée est de choisir le sous-arbre le plus lourd parmi l'ensemble de tous les sous-arbres possibles du DAG. Ensuite, l'algorithme permet de décider si la racine de l'arbre est incluse dans la chaîne principale. Reprenons l'exemple du schéma 1.6 : le bloc b1 a quatre descendants et b2 en a six, donc b2 est choisi. Ensuite b4 a un seul descendant, b5 en a 3. Ce processus est répété et b8 est choisi puis b12. D'après le protocole GHOST, la chaîne principale serait $b2 \leftarrow b5 \leftarrow b8 \leftarrow b12$. Avec cet algorithme, une nouvelle notion est ajouté : bloc oncle (*uncle* ou parfois aussi *ommer*). Pour un bloc donné, il s'agit des autres blocs qui partagent le même parent. Dans Bitcoin, ce sont des blocs orphelins, mais dans le cadre de GHOST ceux-ci sont valorisés même si ce ne sont pas leurs mineurs qui ont résolu le crypto-puzzle le plus rapidement (et que leur transaction ne sont pas comptabilisés). Initialement, Ethereum devait implanter ce protocole, mais l'implantation d'Ethereum a beaucoup évolué depuis 2015 et finalement une version hybride entre la règle de Bitcoin et GHOST est implantée : c'est la branche qui comptabilise la difficulté totale la plus grande [9] qui est sélectionnée.

Plus généralement, sur les mécanismes de consensus, Zhang et al. [ZP19] ont montré qu'aucun des protocoles basé sur le consensus de Nakamoto (basé sur la preuve de travail) ne protège complètement contre les trois attaques suivantes simultanément : double dépenses, *selfish mining* (voir section 2.2.1) et *feather-forking* (voir section 2.2.1).

Un autre élément de comparaison entre Bitcoin et Ethereum est le temps entre chaque nou-

9. dit autrement, la somme des difficultés des blocs la plus grande

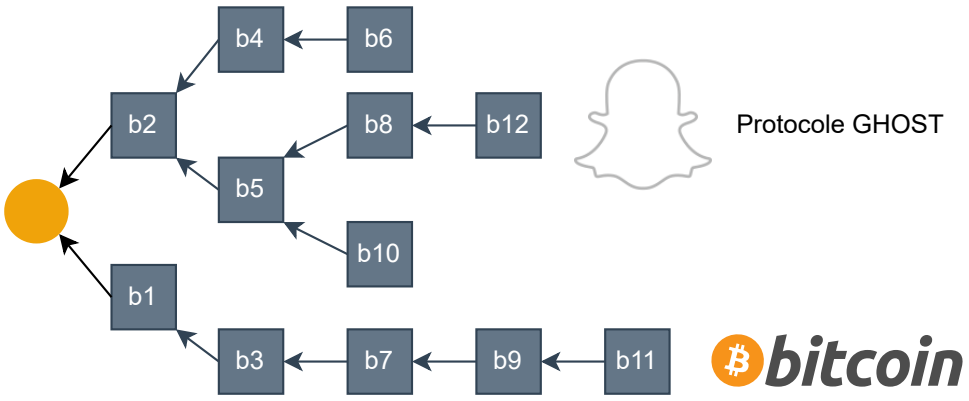

FIGURE 1.6 – Représentation schématique de la règle de sélection de la chaîne dite *longest chain rule* pour Bitcoin, et le protocole GHOST dont une variante est utilisée par Ethereum. Schéma inspiré du MOOC « Blockchain Scalability and its Foundations in Distributed Systems » de Vincent Gramoli [Gra].

veau bloc et la durée de l'ajustement de la difficulté de minage. Bitcoin maintient un temps entre deux blocs de 10 minutes environ et ajuste la difficulté de minage en conséquence tous les 2016 blocs (2 semaines). Ethereum maintient un temps de 15 secondes entre deux blocs et ajuste la difficulté à chaque bloc (en PoW).

1.3.5 Compromis entre vitesse des transactions et robustesse

La notion de scalabilité d'une blockchain est liée à la vitesse à laquelle les transactions sont entérinées. Cette dernière est elle-même lié à la vitesse à laquelle les blocs sont confirmés au sein du réseau. On dérive le nombre de transactions par seconde en fonction du nombre de blocs confirmés par unité de temps (et du nombre de transactions incluses dans ces blocs). Les développeurs de Bitcoin annoncent que le système peut atteindre 7 transactions par seconde [10] et Ethereum, quant à elle, annonce que le système peut atteindre jusqu'à 45 transactions par seconde. Dans la pratique, les différentes études sur ce sujet [SZ15] [CMVM18] [DD19] [BFV19], montrent plutôt qu'en pratique le débit des transactions de Bitcoin et Ethereum est moindre, à savoir entre 3 et 7 transactions par seconde pour Bitcoin et environ 10 transactions par seconde pour Ethereum

La notion du nombre de blocs de confirmation à attendre avant de considérer une transaction entérinée est fondamentale comme nous l'avons vu dans la section 1.1. Pour Bitcoin, un service de paiement considère qu'un maximum de 6 blocs de confirmation (1h d'attente) peuvent être nécessaires pour la plupart des transactions. Pour Ethereum, ce nombre est plutôt de 12 blocs de confirmation (3 minutes d'attente) pour la plupart des transactions [11]. Cependant, les auteurs de [GKW+16], ont montré que pour le même niveau de robustesse que Bitcoin, Ethereum (en PoW) a besoin de 37 blocs de confirmation, étant donné un attaquant qui possède 30% de la puissance de minage. Plus généralement, ils ont montré qu'une blockchain utilisant la preuve de travail ne peut pas dépasser 60 transactions par seconde sans affecter la robustesse du système.

10. https://en.bitcoin.it/w/index.php?title=Scalability&oldid=68329
11. https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times

1.3.6 Sélection des voisins et propagation des messages

Pour établir une connexion, un nœud sélectionne des pairs potentiels depuis sa liste de contacts selon certains critères. Dans le réseau P2P de Bitcoin, la stratégie de sélection des voisins est basé sur l'aléatoire [DW13] [SN18]. Pour Ethereum, en théorie avec l'utilisation d'une DHT, la stratégie de sélection des voisins devrait être différente de celle de Bitcoin, mais nous avons remarqué qu'en pratique, elle est basée également sur l'aléatoire (voir chapitre 4 pour de plus amples détails). Ethereum construit ainsi une seconde topologie P2P non structurée, semblable à Bitcoin, pour la propagation des messages. Les services offerts par la DHT ne sont pas utilisés en pratique, car il n'y a pas de stockage ni de recherche de ressources dans le réseau, l'ensemble des pairs possède toutes les ressources, ici les blocs.

Conclusion

Les blockchains Bitcoin et Ethereum semblent de prime abord très différentes, on remarque qu'elles partagent beaucoup de points communs, notamment en termes de mécanismes de consensus et de stratégie de diffusion de l'information par une topologie non structurée. Dans le tableau 1.1, nous présentons un tableau de synthèse des caractéristiques des deux blockchains publiques. Dans les chapitres 3 et 4, nous étudions en détail les caractéristiques des réseaux P2P de Bitcoin et Ethereum à l'aide de techniques de supervision afin d'analyser plus en profondeur les similarités ou différences de ces deux réseaux.

Le chapitre suivant s'intéresse à la sécurité des réseaux P2P de blockchain et à leur impact sur la sécurité du système.

12. Plus d'ajustement après avoir atteint la *Terminal Total Difficulty* juste avant de passer au PoS, le 15 septembre 2022

TABLE 1.1 – Tableau de synthèse des caractéristiques de Bitcoin et Ethereum.

	Bitcoin	**Ethereum**
Mécanisme de consensus	PoW	PoW (PoS après le 15 septembre 2022)
	Règle de la chaîne la plus longue	Chaîne avec la somme des difficultés des blocs la plus grande (règle dérivée de `GHOST`)
Temps moyen entre blocs	10 minutes	15 secondes
Ajustement de la difficulté	Après 2016 blocs ($\sim$ 2 semaines)	À chaque nouveau bloc [12]
Architecture P2P	Non structuré	Structuré (DHT) et non structuré
Sélection des voisins	Aléatoire	
Propagation des messages	Par inondation	
Scalabilité (transactions/s)	$<$10	$<$45
Crypto-monnaies en circulation	Plus de 19 000 000 BTC (maximum 21 000 000)	Plus de 122 000 000 ETH (pas de limite)
Halving	Tous les quatre ans (récompense 6,25 BTC par bloc + frais des transactions)	Pas de *halving* (récompense 2 ETH par bloc + frais de transactions, en PoW)
Smart contracts	$\sim$ (Langage Script non Turing-complet)	✓ (Langage Solidity Turing-complet)

Chapitre 2

Sécurité dans les réseaux pair-à-pair de blockchains

Sommaire

Introduction

La fiabilité d'un réseau P2P est primordiale, car celui-ci est la brique de base sur laquelle repose les protocoles de plus haut niveau garantissant les propriétés du système distribué. Les blockchains publiques ne dérogent pas à la règle : les réseaux P2P sous-jacents sont très souvent mis à l'épreuve, tant par les chercheurs en sécurité que par les utilisateurs malveillants. Le problème le plus basique que l'on peut rencontrer avec un système basé sur une architecture pair-à-pair publique dérive de l'autonomie des pairs qui constituent le réseau P2P. Ceux-ci doivent suivre les protocoles établis pour garantir le bon fonctionnement du système, mais ils peuvent, égoïstement, trouver un plus grand intérêt à violer le protocole et dans ce cas rendre plus vulnérable le système dans son ensemble.

En outre, il existe des attaques qui exploitent des failles plus complexes dont les conséquences peuvent être très critiques. Plusieurs études proposent une synthèse des problèmes de sécurité que l'on peut trouver dans les systèmes de blockchains [LJC+17] [CSKLR18], mais dans cette thèse, nous nous sommes concentrés sur les problèmes impliquant le réseau P2P. Nous pouvons les classifier en deux catégories : les attaques qui visent directement le réseau P2P (notamment via des attaques Sybil ou Eclipse) ou plus généralement les communications, et celles pour lesquelles le réseau n'est pas la cible, mais peut être un levier pour faciliter d'autres attaques au niveau applicatif.

D'autres attaques n'impliquent pas les communications. Par exemple, il existe également des attaques qui ciblent directement l'implantation et les primitives des smart contracts. Nous pouvons citer l'exemple célèbre du vol d'ethers du projet « The DAO » dans lequel les attaquants ont exploité une faille dans les appels récursifs du smart contract du projet pour dérober des Ethers. Comme elles l'ont montré, ces attaques ont de fort impacts, mais dans le cadre de cette thèse, nous ne nous sommes pas intéressés à la sécurité des smart contracts ou du mécanisme de consensus.

Dans la suite de ce chapitre, nous allons présenter ces problèmes de sécurité, en commençant par deux attaques classiques qui ciblent les réseaux d'overlay : l'attaque Sybil et l'attaque Eclipse. Le partitionnement du réseau est l'un des objectifs de ces deux attaques dans le contexte des blockchains. Nous verrons aussi qu'un réseau P2P peut aussi être partitionné par l'exploitation de failles intrinsèques au protocole de routage de l'Internet. Nous verrons ensuite comment ces attaques sont utilisées pour mener des attaques plus spécifiques et dont les conséquences sur les blockchains ciblées peuvent être sévères. Pour finir, nous verrons quels sont les mécanismes de défense possibles contre ces attaques et quelles contremesures sont effectivement implantées dans Bitcoin et Ethereum.

2.1 Attaques des communications

2.1.1 Attaques via le réseau P2P

La sécurité d'un réseau P2P repose entre autres sur l'indépendance des pairs qui le composent, comme nous l'avons vu dans le chapitre 1. Deux attaques exploitent ce manque d'indépendance couplé parfois à une perturbation des communications et servent de leviers pour cibler des composants clés des systèmes attaqués. Il s'agit de l'attaque Sybil et de l'attaque Eclipse.

Attaque Sybil

John R. Douceur décrit en 2002 le principe de l'attaque Sybil [Dou02] qui exploite justement le faible degré d'indépendance entre les pairs. En effet, le principe pour un attaquant est d'insérer un grand nombre de nœuds dans le réseau. Ces nœuds ne sont donc pas indépendants et l'attaquant peut représenter une part non négligeable du réseau. Ainsi, lorsque ce principe d'indépendance est nécessaire, par exemple dans le cas des protocoles de mixing (nous abordons ce cas applicatif plus loin), les données partagées peuvent être reçues uniquement par des nœuds Sybils. La faiblesse des réseaux P2P face à cette attaque découle directement de la difficulté de génération d'une identité sur le réseau. Dans l'état de l'art des réseaux P2P, cette attaque est particulièrement dévastatrice dans le cadre des réseaux P2P publics qui n'exercent pas de contrôle d'accès sur les pairs qui peuvent librement rejoindre le réseau. En outre, les réseaux P2P structurés où les pairs possèdent des IDs réseau et utilisent une table de hachage distribuée sont particulièrement vulnérables, car l'attaquant peut ainsi cibler la partie du réseau dont il souhaite prendre le contrôle en générant et sélectionnant des identifiants appropriés comme ce fut plusieurs fois démontré sur le réseau KAD [LNR06, SEB07, KLR09, CCF10] et illustré par la figure 2.1.

C'est aussi le cas pour Ethereum où un pair est identifié par la clé publique qu'il génère et cette identité est placée dans la DHT. Nous verrons dans la suite de ce chapitre la vulnérabilité d'Ethereum face aux attaques Sybil et les recenserons plus particulièrement dans le chapitre 6. A contrario, dans le cas de Bitcoin qui repose sur un réseau P2P non structuré, l'identité d'un pair réside dans le couple adresse IP–port et naïvement, il peut sembler difficile de forger ou usurper une adresse IP, mais les travaux de Heilman et al. [HKZG15] montre que c'est tout à

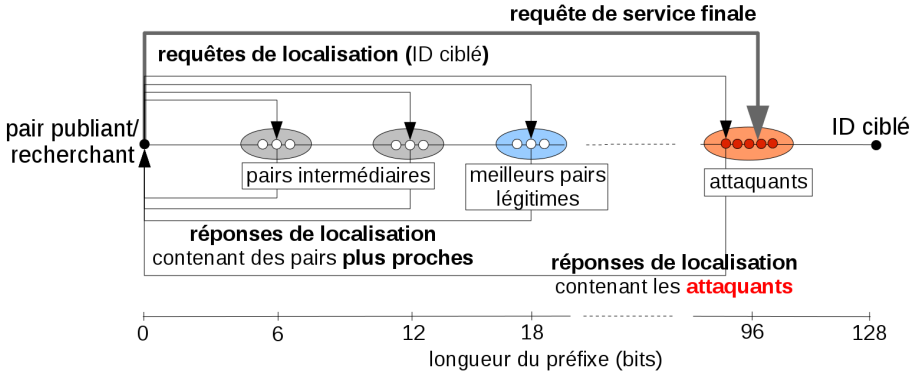

FIGURE 2.1 – Schéma d'une attaque Sybil sur la DHT de KAD prenant le contrôle de l'indexation du contenu cible (d'après [Cho11])

fait possible, bien que coûteux. Il est beaucoup plus facile de créer des Sybils en changeant les numéros de ports parmi les 64500 possibles et cela est suffisant pour mener une attaque Sybil d'envergure pouvant largement dépasser le nombre de pairs légitimes avec une seule adresse IP. Une telle attaque est très peu discrète et très simple à prévenir, car il suffit pour un nœud de vérifier que ses connexions ne sont pas monopolisées par des pairs avec une unique adresse IP (mais différents ports).

En outre, Neudecker et al. [Neu19] ont montré que des attaques Sybils ont eu lieu sur le réseau P2P de Bitcoin par le passé. Ils n'ont cependant pas analysé les intérêts d'une telle attaque et supposent que cela peut être dû à une mauvaise configuration. Également, en 2021, les auteurs de l'étude de synthèse [IM21] ont proposé un bilan des risques sur les blockchains liés aux attaques Sybils et doubles dépenses.

Attaque Eclipse

L'attaque « Eclipse » est très classique au sein des réseaux d'overlay et l'attaquant utilise couramment une attaque Sybil préliminaire pour faciliter et amplifier cette attaque. Les auteurs de l'article [SNDW06] décrivent comment une telle attaque est mise en œuvre et comment s'en prémunir. Le but recherché par l'attaquant est de monopoliser les connexions d'un ou plusieurs pairs corrects (c'est-à-dire qui suivent les protocoles) avec ses nœuds malveillants. Ainsi, il isole complètement les victimes du trafic du réseau P2P et peut les tromper avec les messages qu'il aura forgés. On dit qu'il « éclipse » le trafic du réseau d'overlay vu par les victimes de l'attaque. Les réseaux P2P structurés sont également vulnérables à cette attaque [CDG+03]. Elle consiste alors à remplir la table de routage du nœud victime avec des Sybils, certaines études étant allées jusqu'à créer une partition au sein du réseau P2P KAD [WTC+08]. Appliquée maintenant à un réseau P2P d'une blockchain, l'idée générale d'une telle attaque, décrite par Heilman et al. [HKZG15] dans leur article sur la mise en œuvre d'une attaque Eclipse sur Bitcoin, est de réussir à remplir la liste de contacts d'un nœud cible avec des nœuds malveillants afin que la victime établisse l'ensemble de ses connexions sortantes vers lesdits nœuds. L'attaquant peut alors remplir l'ensemble des connexions entrantes de la cible avec ses nœuds et la victime ne recevra dès lors plus que le trafic transmis par les nœuds de l'attaquant. La Figure 2.2 schématise l'état d'un réseau P2P dont un des nœuds est victime d'une attaque Eclipse. Ce nœud a établi toutes ses connexions sortantes vers les nœuds de l'attaquant et celui-ci a monopolisé l'ensemble des connexions entrantes de la victime avec ses nœuds malveillants. Un travail similaire a été produit

sur Ethereum par les auteurs de l'article [WG16].

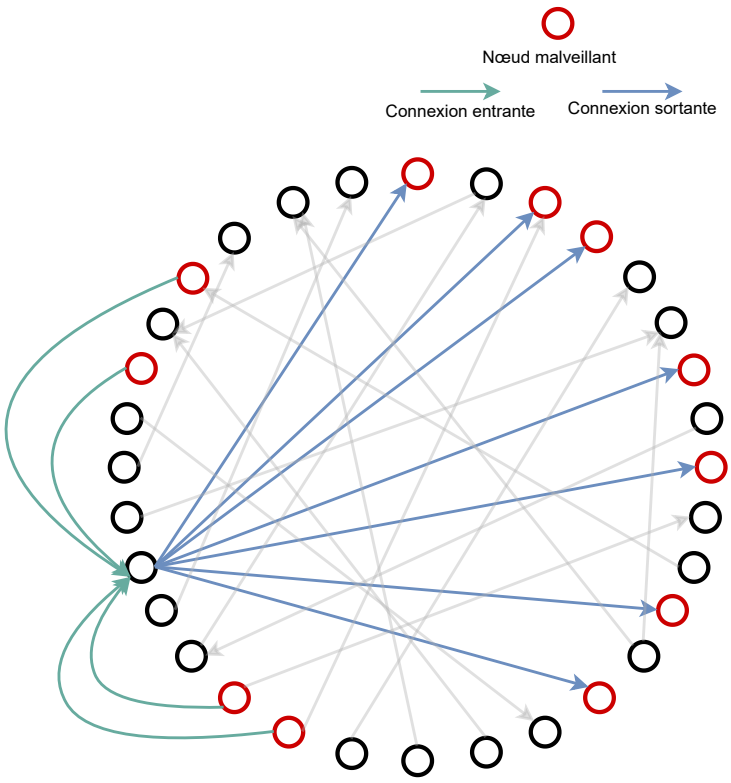

FIGURE 2.2 – Schéma d'une attaque Eclipse réussie.

Dans le cadre d'une DHT un autre type d'attaque éclipse ciblant les contenus indexés plutôt que les pairs peut passer par le contrôle du système d'indexation plutôt que par la monopolisation des liens d'un pair [LNR06, SEB07, KLR09, CCF10], comme illustré par la figure 2.1. Étant donné le caractère déterministe du routage, l'insertion d'un petit nombre de nœuds Sybil (une vingtaine pour KAD [CCF10], mais ce chiffre varie en fonction du taux de réplication spécifique à chaque DHT) au plus proche des données ciblées suffit pour prendre le contrôle de son indexation et en faire disparaître des données. Les mêmes adresses IP pouvant être réutilisées pour déployer des attaques en différents points de l'espace d'adressage.

Comme nous l'avons évoqué précédemment, l'attaquant peut tirer parti d'une attaque Sybil pour mener à bien une attaque Eclipse, comme le montre les auteurs de l'article [MHG18]. En effet, il peut utiliser les nombreuses identités (nœuds sybils) qu'il a créé pour remplir la liste de contacts d'une cible et augmenter ses probabilités de réussir une attaque Eclipse. Dans les travaux de [MHG18], il s'agit d'une attaque menée sur le réseau Ethereum, car les nœuds sont identifiés par un ID réseau tandis que sur le réseau P2P de Bitcoin les nœuds sont identifiés par leur adresse IP (et numéro de port). Les auteurs ont montré qu'il était beaucoup plus facile et moins coûteux de mener une attaque Eclipse (à l'aide de nœuds sybils) sur Ethereum que sur Bitcoin, car la génération d'ID réseau (clé publique ECDSA pour Ethereum) est triviale alors que posséder ou usurper des centaines ou milliers d'adresses IP (pour attaquer un nœud Bitcoin) est bien plus difficile et complexe à mettre en œuvre.

Dans la section 2.2.1, nous expliciterons les attaques de plus haut niveau visant les blockchains publiques qui peuvent bénéficier de l'attaque préalable du réseau P2P. Nous aborderons entre autres, les cas de l'attaque du *Selfish Mining* et l'attaque *Balance*.

2.1.2 Attaques via le protocole de routage de l'Internet, BGP

Nous avons vu que les deux attaques précédentes utilisent les protocoles de la couche applicative des réseaux P2P de blockchains pour mener des attaques de partitionnement. Comme le montre Apostolaki et al. [AZV17], un tel objectif est aussi atteignable en exploitant une vulnérabilité intrinsèque de BGP, le protocole de routage de l'Internet.

Le rôle de ce protocole est d'interconnecter les *autonomous systems* (AS) qui sont responsables d'un ensemble d'adresses IP identifié par un préfixe. En pratique, ce protocole permet de contrôler comment les paquets IP sont transmis sur l'Internet par l'échange d'annonces de routes (associées à des préfixes IP) entre AS. Pour un préfixe IP, l'AS d'origine est responsable d'annoncer la route aux autres AS pour permettre la bonne transmission des paquets à ces adresses IP. Mais les routeurs BGP ne vérifient pas la validité des routes annoncées et cela permet alors à un AS malveillant d'attirer à lui tout le trafic destiné à un préfixe IP en annonçant être responsable d'un préfixe plus spécifique. Le routage IP privilégiant les routes plus spécifiques (*longest match*), ce seront celles-ci qui seront utilisées. On appelle cela un détournement d'adresses IP (*IP hijacking* en anglais).

Ces détournements arrivent très souvent et sont rapidement résolus, car ils sont, pour la plupart, issus d'une mauvaise configuration de la part des AS et sont très visibles par les outils de supervision de l'Internet (un simple *traceroute* permet de se rendre compte qu'il y a un problème). Un exemple très célèbre, car il a touché un service très populaire sur l'Internet, concerne l'AS Pakistan Telecom et Youtube. Le gouvernement de Pakistan a annoncé en 2008 un blocage de la plateforme Youtube sur son territoire et Pakistan Telecom, chargé de l'exécution de ce blocage, a annoncé un préfixe plus spécifique pour le trafic à destination de Youtube entrainant l'inaccessibilité de la plateforme pour le monde entier. Quelques heures plus tard le problème avait été résolu.

Concernant Bitcoin, les auteurs de [AZV17] ont montré à travers une méthodologie très détaillée comment un détournement d'adresses IP permet de partitionner le réseau P2P. Ce partitionnement a un impact très fort, car le trafic Bitcoin n'est pas chiffré. Parmi les résultats les plus impressionnants, le détournement de moins de 100 préfixes permet d'isoler environ 50% de la puissance de calcul de Bitcoin. Ce détournement est également très rapide (moins de 2 minutes) et il peut se passer plusieurs heures avant de retrouver une densité de connexion proche de celle avant l'attaque. Comme pour tous les détournements d'adresses IP, celui décrit dans ces travaux est très visible et n'est pas à la portée de n'importe quel attaquant : ce ne sont que les grandes infrastructures de l'Internet, du niveau d'un AS, qui en ont les capacités. Cela explique pourquoi cette attaque, bien que très puissante, n'a vraisemblablement pas été exploitée à grande échelle dans Bitcoin.

Une attaque qui utilise le même modèle d'attaquant (du niveau d'un AS) mais qui ne tire pas parti d'une manipulation du routage a été décrite dans [TCM+20]. Les auteurs présentent EREBUS, une attaque discrète, dont le principe réside dans l'utilisation de ce qu'ils appellent des *shadow IP*. Il s'agit d'adresses IP sur lesquelles un nœud Bitcoin n'est pas forcément hébergé, voire pas rattaché à un quelconque hôte. Un AS intermédiaire entre un AS qui héberge des nœuds Bitcoin et un AS qui héberge ces *shadow IP* peut forger des messages à destination des nœuds Bitcoin pour faire croire à des connexions légitimes et intercepter ensuite les réponses. Les AS capables d'une telle attaque ne sont que les plus importants et d'après les évaluations des auteurs, sa mise en place prend environ 5 à 6 semaines.

2.2 Application à la sécurité de la blockchain

Outre les attaques spécifiques aux réseaux P2P, les blockchains publiques peuvent également être victimes d'attaques qui ciblent d'autres caractéristiques, notamment les protocoles et algorithmes responsables du consensus, mais également les primitives des smart contracts par exemple. Les objectifs de telles attaques se situent au niveau applicatif, allant du déni de service au vol ou détournement de crypto-monnaies. Nous verrons que souvent ces attaques peuvent tirer parti de la perturbation des communications pour amplifier leur impact.

2.2.1 Attaques liées au mécanisme de consensus et comportement P2P

Dans le cadre des blockchains publiques dont le mécanisme de consensus est basé sur la preuve de travail, les chercheurs ont réussi à mettre en exergue des faiblesses qu'un attaquant peut exploiter à des fins malveillantes (gain d'influence, vol de crypto-monnaies, etc). L'attaque Eclipse (conjointe à une attaque Sybil) fournit un important levier à un attaquant pour mener des attaques dont les conséquences sont multiples :

— réduire la puissance de calcul globale des mineurs (attaque 51% [Nak08], voir section 1.1) ;
— permettre l'attaque *Balance* (voir section 2.2.1) ;
— permettre une double dépense (*double spending*) avec transaction à 0 (voire N sous conditions) blocs de confirmation [KAC12].

En effet, les doubles dépenses peuvent être facilitées par une attaque Eclipse, même avec des transactions entérinées après n blocs (Nombre standard de nouveaux blocs attendus pour considérer une transaction comme « irréversible »). Schématiquement, un nœud éclipsé ne voit pas le trafic normal des blocs sur le réseau et donc peut être trompé par les blocs que l'attaquant lui-même aura minés, mais qui ne sont pas considérés dans la chaîne principale de la blockchain par les autres nœuds du réseau.

Attaque du minage égoïste (*Selfish Mining* et *Stubborn Mining*)

Nakamoto, dans le papier qui décrit Bitcoin [Nak08], a théorisé une attaque où un ensemble de mineurs qui arrive à rassembler 51% minimum de la puissance de calcul totale du réseau gagne le contrôle sur la blockchain, car la branche principale de celle-ci sera composée en majorité des blocs de ces mineurs. Eyal et Sirer [ES14] ont montré en 2014 qu'une coalition de mineurs malhonnête qui représente moins de 51% peut gagner davantage de récompense de minage en adoptant une stratégie dite de « minage égoïste » (*Selfish mining* en anglais). Cette stratégie découle directement du principe de base dans le paradigme pair-à-pair qui dicte qu'un utilisateur sera tenté de dériver des protocoles si cela lui offre un intérêt plus grand que d'agir honnêtement dans le réseau P2P.

L'idée derrière cette stratégie est de faire en sorte que les mineurs honnêtes gâchent leur puissance de calcul en minant des blocs sur une branche de la blockchain qui finalement ne sera pas prise en compte. Pour cela, la coalition de mineurs égoïstes mine des blocs qu'elle ne révèle pas publiquement, en d'autres termes, elle mine sur une branche privée de la blockchain. Plusieurs événements peuvent alors se passer :

— la branche publique dépasse la taille de la branche privée, alors la coalition arrête de miner sur sa branche privée et adopte la branche publique ;
— la branche publique rejoint la taille de la branche privée (après avoir été en dessous), alors la coalition rend publique sa propre branche et elles entrent en concurrence et l'une ou l'autre peut gagner ;

— la branche privée reste toujours plus grande que la branche publique, alors la coalition conserve son avance et publie régulièrement les blocs de sa branche pour récupérer les récompenses.

Cette stratégie prend en compte deux paramètres, α la puissance de calcul relative de la coalition (entre 0 et 0,5) et γ la fraction du réseau qui suit les blocs de la coalition malhonnête (entre 0 et 1). En prenant des valeurs réalistes de ces deux paramètres, les auteurs ont montré qu'avec $\gamma = \frac{1}{2}$ la stratégie de *Selfish Mining* (dans le cas où la branche privée est toujours en avance) est plus profitable que le minage honnête pour $\alpha = \frac{1}{4}$. Il déclare également que cette attaque est faisable et rentable pour toutes les valeurs de α inférieures à 0.5 même les plus petites.

Les travaux de Grunspan et al. [GP18a] approfondissent l'étude d'Eyal et Sirer, notamment sur l'analyse temporelle de cette attaque et sur une mesure plus exacte de la profitabilité. Ils montrent que la stratégie de *Selfish Mining* devient profitable uniquement après l'ajustement de la difficulté de minage (après 2016 blocs sur Bitcoin donc deux semaines). La durée minimum de l'attaque pour être plus rentable que le minage honnête est de 23,8 jours avec $\gamma = \frac{1}{2}$ et $\alpha = 0,43$. Cette puissance de calcul de 43% de la puissance totale est très proche des 51%. Pour une valeur plus réaliste de 10%, et $\gamma = 0,9$ alors la durée minimum de l'attaque pour être rentable est de 10 semaines. Cette durée atteint presque deux années dans le cas où $\alpha = 0,01$ et $\gamma = 0,99$. Cette étude montre que l'attaque de *Selfish Mining* reste possible et rentable, mais peu vraisemblable à observer en vrai, car assez coûteuse en puissance de calcul ou en temps d'exécution avant rentabilité. De plus, depuis les travaux d'Eyal et al. en 2014, plusieurs services de supervision [13] ont été utilisés par le passé pour scruter activement les agissements des pools de mineurs et vérifier si cette attaque était en train d'avoir lieu ou si elle a déjà eu lieu dans le passé.

L'attaque du *Selfish Mining* peut aussi être combinée à une attaque Eclipse et être étendue en ajoutant des scénarios plus complexes pour augmenter les cas où la coalition de mineurs malhonnête est rentable. Il s'agit des travaux de Nayak et al. [NKMS16] dans lequel ils nomment cette attaque *Stubborn Mining* (en français, le minage acharné ou obstiné). L'attaque Eclipse peut être utilisée pour nullifier la puissance de calcul d'un ensemble de mineurs, ou alors pour forcer une coopérative de mineurs à suivre la stratégie de l'attaquant. Grunspan et al. [GP18b] [GP18c] ont également montré que les stratégies introduites par le *Stubborn Mining* sont plus profitables que le *Selfish Mining* pour des petites valeurs de α (pour rappel, il s'agit de la puissance de calcul relative de la coalition malhonnête).

Une autre attaque qui tire profit d'un comportement déviant des mineurs a été décrite en 2013 par Andrew Miller [Mil13] et appelé *feather-forking*. Un *feather-fork* consiste, pour un mineur dont la puissance de calcul est inférieure à 51%, à refuser de miner de nouveaux blocs sur une branche de la blockchain dans laquelle une transaction d'un utilisateur cible y figure. En d'autres termes, il s'agit d'une forme de déni de service à l'encontre d'un utilisateur pour empêcher que ses transactions soient contenues dans les blocs de la chaîne principale.

Attaque « *Balance* »

Précédemment, nous avons présenté les travaux qui étudient le *Selfish Mining*, qui dans sa forme classique requiert tout de même une importante part de la puissance de calcul de minage du réseau (mais inférieure à 51% tout de même) pour permettre à un attaquant de gagner plus que ce qu'il aurait gagné en étant honnête. En 2017, les auteurs de l'article [NG17] ont montré qu'un attaquant peut tirer avantage d'une attaque Eclipse et ajouter des délais dans les messages du réseau P2P pour faire des doubles dépenses sans détenir une importante part de

13. `http://organofcorti.blogspot.com/` (mais il n'y a plus de données depuis 2017) et coinometrics.com (mais ce service n'est plus en ligne en 2022)

la puissance de calcul de minage (bien moins important que pour le *Selfish Mining*). En effet, le principe de l'attaque consiste à former, à partir du graphe du réseau P2P complet, deux sous-graphes de puissance de calcul de minage équivalente (d'où le nom de l'attaque, *balance* en anglais signifie « équilibre ») et introduire des délais dans les messages qui arrivent dans un des sous-graphes. Le schéma 2.3 illustre le fonctionnement de l'attaque. L'attaquant peut émettre une transaction qu'il ne propagera que dans un des deux sous-graphes (celui du haut dans le schéma 2.3). À cause des délais, les mineurs créent deux branches concurrentes (*fork*) dans la blockchain et l'attaquant peut alors augmenter le poids du sous-graphe dans lequel il n'a pas propagé sa transaction, en ajoutant sa propre puissance de calcul à celui-ci, et à terme faire gagner la branche de la blockchain correspondante (celle du bas dans le schéma). Tant que sa transaction n'est pas confirmée (après 6 nouveaux blocs sur Bitcoin par exemple), il continue le partitionnement du réseau. Quand il aura reçu les biens en échange des jetons de sa transaction, il arrête le partitionnement (suppression des délais) et lorsque l'ensemble du réseau est à nouveau synchronisé, la transaction que l'attaquant a émise ne sera pas valide, car elle appartient à la branche de la blockchain qui n'est pas retenue par le réseau. L'attaquant aura donc obtenu des biens ou marchandises en échange de jetons que le marchand ne récupérera finalement pas et l'attaquant pourra les dépenser à nouveau.

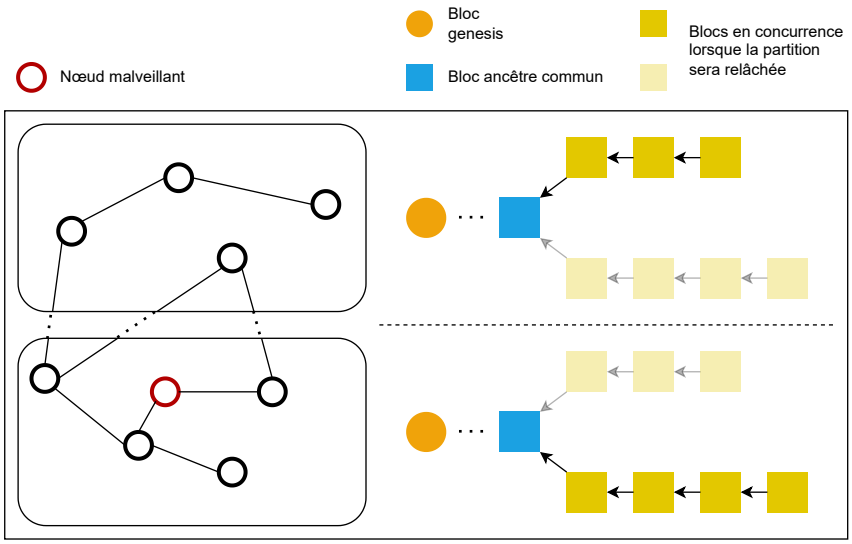

FIGURE 2.3 – Schéma d'une attaque *Balance*. Ici, nous nous plaçons dans le cas de la blockchain Bitcoin et l'attaquant envoie sa transaction dans le sous-graphe du haut et mine dans celui du bas.

L'introduction de délais dans les messages peut être accomplie de diverses manières. Le partitionnement du réseau en deux sous-graphes de puissance de calcul équivalente peut être obtenu par une attaque Eclipse de grande ampleur et les nœuds malveillants utilisés peuvent alors servir à ne propager les transactions que dans un sous-graphe ou à les propager après un certain délai. Un attaquant peut aussi utiliser une attaque sur les couches protocolaires plus basses, à savoir le protocole de routage de l'Internet BGP (nous avons présenté cette attaque à la section 2.1.2), pour introduire des délais.

Les auteurs ont montré que cette attaque est possible sur les blockchains qui permettent les forks et donc celles basées sur le Proof-of-Work comme Bitcoin et Ethereum. Ils ne donnent pas de moyens de se prémunir contre une telle attaque, mais comme celle-ci nécessite une attaque

Eclipse de grande ampleur couplée éventuellement à une attaque Sybil, on peut se prémunir d'une attaque *Balance* en rendant inefficaces ou très couteuses les attaques Eclipse.

En 2019, Zhang et al. [ZL19], ont proposé une attaque très similaire à l'attaque *Balance* dont la différence réside dans le modèle d'attaque, notamment l'utilisation d'une attaque Sybil préliminaire pour l'introduction de délais dans les messages, ainsi que dans la puissance de calcul nécessaire pour l'attaquant, 32% du total en l'occurrence. On peut observer, encore une fois, qu'une attaque du réseau P2P (ici Sybil) fournit un levier d'attaque pour exécuter une double dépense.

2.2.2 Déni de service (DOS) et désanonymisation

DOS

Chaganti et al. [CBR+22] proposent une étude de synthèse sur les risques de déni de service dans les blockchains et l'attaque Sybil et Eclipse sont des vecteurs directs qui mènent vers des attaques de déni de service. En effet, en cas d'attaque Eclipse réussie, les connexions de la victime sont monopolisés par les nœuds (sybils) de l'attaquant et il impose ainsi la vue de la blockchain qu'il souhaite. Il peut décider d'empêcher certaines transactions ou blocs de transiter jusqu'à la victime et donc la priver de la vue de l'état de la blockchain. Également, il peut l'empêcher d'émettre ses transactions et donc d'utiliser un service de paiement ou, si la victime participe au minage, de l'empêcher d'émettre ses blocs et donc de recueillir les récompenses associées le cas échéant. Il s'agit de cas d'applications très localisés, mais très puissants contre une cible de choix.

Protocole de « mixing »

Une des autres applications directes de l'attaque Sybil, plus globale que la précédente, sur les réseaux P2P de blockchains concerne les protocoles de mixing [BNM+14]. Ces protocoles permettent d'ajouter une couche d'anonymisation sur les transactions et notamment d'empêcher de retrouver à qui un utilisateur envoie une transaction, car Bitcoin, par exemple, a été extensivement étudié sur son prétendu anonymat [AKR+13] [MPJ+13] [RS13] et les chercheurs ont montré qu'il était possible d'établir des liens entre les utilisateurs, les adresses et les transactions. Il existe deux familles de système de mixing (on parle aussi de *tumblers*) : centralisé et décentralisé. Les systèmes centralisés reposent sur un acteur tierce partie qui centralise les transactions à mélanger et les redistribue. Cette solution n'est pas plébiscitée par la communauté, car elle possède toutes les faiblesses d'un système centralisé : les coûts sont importants, une forte confiance envers la tierce partie est nécessaire, et elle constitue un point de défaillance unique. Les systèmes décentralisés reposent sur des protocoles tels que CoinSwap, CoinJoin, ou SharedCoin par exemple. Bissias et al. [BOLL14] ont analysé les protocoles centralisés et décentralisés et ont montré qu'ils sont vulnérables à une attaque Sybil. Un attaquant peut « défaire le mélange » des transactions et inférer les liens entre utilisateurs (concernant les transactions) en insérant des nœuds sybils intelligemment dans le réseau. Les auteurs proposent un nouveau protocole appelé Xim résistant à cette attaque. Mikerah Quintyne-Collins [Qui19] propose également une caractérisation des attaques Sybils ciblant les protocoles de mixing, en identifiant deux risques sur les protocoles : le risque de la désanonymisation des transactions et le risque de déni de service.

2.3 Moyens de défense

2.3.1 Contre les attaques Sybil et Eclipse

Solutions historiques des réseaux P2P

Depuis la théorisation de l'attaque Sybil divers mécanismes de défense ont été imaginés et implantés dans plusieurs systèmes distribués. Les études [LSM06] et [UPS11] recensent la plupart d'entre eux. On peut distinguer trois catégories : les règles préventives, le contrôle d'accès et les contraintes structurelles sur la topologie.

Nous pouvons commencer en citant quelques règles préventives simples implantées dans KAD qui visent à rendre les attaques Sybils plus coûteuses :

— les réponses non sollicitées sont ignorées ;
— le nombre maximum de messages reçus d'un même pair par intervalle de temps est limité ;
— il n'est pas possible d'insérer dans une table de routage plus d'un pair avec la même adresse IP et plus de 10 issus d'un même sous-réseau IP (/24), de plus ces pairs doivent être dans des K-buckets différents (autrement dit assez espacés sur l'espace d'adressage) ;
— l'usurpation d'identité est protégée par un échange de messages applicatifs de type *Three-way handshake*.

Ces règles basiques ont néanmoins été montrées efficaces [CCF09] et augmentent énormément le coût pour un attaquant, bien qu'elles n'empêchent pas des attaques distribuées et davantage ciblées. Celles-ci sont détectées par les auteurs de [CCFD13] qui ont proposé un système complètement distribué basé sur une détection locale des pairs suspects par observation de la distribution des identifiants autour d'un ID cible, et une révocation des pairs qui font dévier de la norme, car considérés comme statistiquement trop proches de la cible. Cette solution est efficace contre des attaques ciblées, mais manque d'une vue globale de la DHT pour corréler les nœuds Sybils présents en différentes places. Elle ne peut détecter si quelques adresses IP génèrent beaucoup de Sybils sur l'ensemble de l'espace d'adressage, et n'empêche donc pas des attaques plus globales. D'autres approches proposées par [CDG+03] et [REMLP07] font écho à la preuve de travail des blockchains. Elles visent à conditionner l'obtention d'un identifiant valide dans le réseau à la résolution d'un puzzle cryptographique et à borner cette validité dans le temps. Cela rend l'obtention de nombreux nœuds Sybils et leur placement précis plus coûteux en ressources calculatoires. Une autre série de solutions [YKGF06] [YGKX08] consiste à demander aux pairs d'afficher des liens sociaux entre eux afin d'identifier les clusters suspects pouvant conduire à une attaque Sybil. Cette approche, bien qu'efficace en théorie, semble cependant difficilement applicable et n'a jamais été implantée dans un réseau déployé. De plus elle n'empêche pas de petites attaques ciblées.

Les auteurs de [FMR+09] [AMRS08] proposent un contrôle d'accès reposant sur une autorité de certification centrale pour valider les pairs qui rejoignent un réseau P2P. C'est une solution efficace, qui a le mérite de pouvoir suivre et analyser toutes les demandes et d'empêcher un pair de pouvoir choisir son identifiant, mais qui possède les faiblesses du paradigme centralisé et n'est donc pas applicable à des blockchains publiques. Bien que beaucoup plus lourdes et complexes à mettre en œuvre, les systèmes à base d'autorité de certification distribuée sont plus adaptés. Dinger et al. [DH06] ont proposé une approche où l'identifiant d'un nouveau pair est calculé de manière déterministe à partir du préfixe de l'adresse IP et validé par le réseau. Les auteurs de [LMT09] proposent de doter le réseau d'une clé secrète suivant un arbre de fragmentation, chaque pair en possédant un fragment en plus de ses propres clés. Un nouveau pair doit obtenir un certificat valide pour sa clé publique par l'approbation d'un certain nombre de pairs proportionnel à la taille du réseau. Une procédure permettant de révoquer les nœuds malveillants est également

décrite [LMVTT08]. Ces solutions sont cependant difficilement applicables à un réseau existant, car elles remettent fondamentalement en cause la manière de générer les identifiants.

Une troisième voie vise à changer la manière de constituer la table de routage ou d'effectuer le routage des paquets dans l'overlay afin de le rendre plus robuste à l'attaque Sybil, au prix d'une perte d'optimalité dans la sélection des pairs. La solution décrite dans [DLKA05] prend en compte les relations d'amorçage entre les pairs, équivalent aux relations sociales de [YKGF06], avant de les sélectionner. L'article [SCDR04] considère quant à lui le degré de connectivité des pairs dans un réseau P2P non structuré afin d'éviter les clusters suspects, ce qui implique de pouvoir en découvrir la topologie. Une autre approche proposée dans [CKS+06] impose la réinitialisation périodique des tables de routage, la fréquence de leur mise à jour et des identifiants éphémères. Le surcoût en messages est cependant très important et il n'est pas fait état du sort des données indexées dans ces conditions instables. Singh et al. [SNDW06] ont décrit plusieurs moyens de défense contre les attaques Eclipse dans les réseaux d'overlay historiques (distribution de contenu, partage de fichiers) :

— utilisation d'une table de routage avec contraintes fortes (*constrained routing table, CRT*) ;
— sélection des voisins basée sur la proximité, c'est-à-dire le délai de réponse (*proximity neighbor selection, PNS*) ;
— supervision du degré de connexions entrantes et refus de se connecter à un nœud dont ce degré est trop élevé.

Ces techniques se montrent efficaces pour rendre plus compliquées et coûteuses les attaques Eclipse, mais elles ont des limitations qui les rendent difficiles à mettre en œuvre dans le cadre des réseaux P2P de blockchain. L'utilisation d'une CRT n'est possible que si le réseau d'overlay est construit de sorte que chaque nœud dispose d'un identifiant réseau unique et non forgeable, ce qui exclut d'office Bitcoin puisqu'il n'y a pas vraiment d'identifiant sur le réseau hormis l'adresse IP des nœuds. Concernant Ethereum, cette technique n'est également pas envisageable car l'ID réseau utilisé (clé publique ECDSA) est assez facilement forgeable car la génération de celui-ci est très rapide et très peu coûteuse. La deuxième technique qui consiste à baser la sélection des voisins sur la proximité, et donc sur le délai de réponse, est un moyen de défense envisageable pour Bitcoin et Ethereum.

Mais dans la pratique, comme nous l'avons vu dans la section 1.2, Bitcoin et Ethereum implantent une sélection aléatoire des voisins. De plus, dans [SNDW06], les auteurs ont montré que la technique est beaucoup moins efficace avec des tailles de réseau d'overlay supérieur à 10 000 nœuds et nous montrerons dans les chapitres 3 et 4 que la taille des deux réseaux P2P sont de cet ordre de grandeur. La dernière technique est basée sur l'hypothèse que l'attaquant souhaite éclipser plusieurs nœuds du réseau. Dans le cadre des réseaux P2P de blockchain, une attaque Eclipse est intéressante à mener dès le premier nœud éclipsé, notamment dans le cadre d'une attaque de double dépense.

Solutions proposées pour les blockchains

Face à la vulnérabilité de Bitcoin et Ethereum aux attaques Eclipse et l'inefficacité des moyens de défenses, Heilman et al. [HKZG15] ainsi que Marcus et al. [MHG18] ont proposé des contre-mesures à implanter respectivement dans les clients Bitcoin et Ethereum. Pour Bitcoin, les contre-mesures se résument à l'augmentation de la taille des tables de contacts (*new* et *tried*) et le remplacement de la sélection des voisins basé sur le caractère récent des contacts (*timestamp-based selection*) par une sélection aléatoire des voisins. Il aurait été intéressant d'évaluer une sélection basée sur la proximité comme le suggérait Singh et al. en 2006. Pour Ethereum, les contre-mesures implantées empêchent un nœud de traiter des messages applicatif `PING` non sollicités

pendant la phase d'amorçage *seeding phase* dans le réseau P2P et donc un attaquant ne peut pas monopoliser entièrement la liste de contact d'un nœud, car un sous-ensemble de cette liste sera toujours issue de la phase d'amorçage.

Pour compléter ces contre-mesures sur Ethereum, Xu et al. [XGS+20] ont proposé un modèle de détection basé sur une classification par forêt d'arbres décisionnels qui est entrainée sur les paquets UDP `PING` utilisés pour mener une attaque Eclipse. Plus de 90% des paquets utilisés par un attaquant sont identifiés correctement, permettant de prévenir une telle attaque. Les travaux de Swathi et al. [SMP19] proposent quant à eux un système de monitoring des blocs émis pour prévenir d'une attaque Sybil. Leur hypothèse est qu'un nœud qui transmet uniquement les blocs d'une même source est potentiellement un nœud Sybil.

Face à l'omniprésence et la puissance de cette attaque, nous proposons dans le chapitre 6 un recensement des nœuds suspectés de sybils dans le réseau P2P d'Ethereum, ainsi qu'un nouveau système de prévention des attaques Sybils alliant les forces des systèmes centralisés et distribués, dans le chapitre 7, permettant la révocation des nœuds détectés comme suspects dans le réseau.

2.3.2 Contre les détournements BGP

Quelques techniques classiques peuvent être mises en place pour se prémunir d'un détournement d'adresses IP, le détecter avant que l'attaquant puisse l'exploiter ou alors le rendre inexploitable. Il est possible par exemple pour un nœud de superviser le *round-trip time* et augmenter son nombre de connexions vers d'autres pairs si ce temps augmente anormalement, ce qui constitue un symptôme classique d'un *BGP hijack*. Le chiffrement des communications du réseau P2P de Bitcoin ainsi que l'introduction de l'authentification des messages sont d'autres moyens de défense qui rendent moins exploitable ce type d'attaque sur le réseau Bitcoin. L'augmentation du nombre de connexions initiées par un nœud est également une solution efficace.

En outre, Apostolaki et al. proposent dans des travaux ultérieurs [AMMV18] un système, nommé SABRE, qui est un réseau mondial de relais qui est responsable de la propagation des blocs. L'apport principal de ces nœuds relais repose dans la sélection des AS qui les hébergent. En effet, en utilisant les propres règles de BGP, on peut spécifiquement empêcher un nœud d'être la cible d'un détournement : ces relais doivent être hébergés au sein de préfixes très spécifiques comme des /24 dont les AS sont directement connectés les uns aux autres (pas de transitivité du routage, pas d'AS intermédiaires). Cela permet de protéger les communications relais-à-relais.

Il est aussi important de noter que ces attaques ne sont pas à la portée de n'importe quel attaquant : seuls les AS les plus importants et les organisations étatiques ont les infrastructures et les moyens nécessaires à ce type d'attaque. Cela les rend très visibles et les coûts d'une telle attaque peuvent être plus importants que les bénéfices.

Conclusion

Pour conclure, nous avons présenté les principaux problèmes de sécurité des systèmes de blockchains qui sont liés à la sécurité du réseau P2P sous-jacent. Nous avons décrit et détaillé le principe de fonctionnement de deux attaques historiques sur les réseaux P2P, à savoir l'attaque Sybil et l'attaque Eclipse. L'état de l'art sur la sécurité des blockchains montre également que d'autres problèmes de sécurité ciblant la blockchain utilisent souvent des attaques Sybil ou Eclipse préalables pour faciliter l'attaque. Les attaques qui perturbent les communications au niveau du réseau P2P sont relativement faciles à mettre en œuvre et peuvent avoir un impact important sur un système comme une blockchain et malgré la diversité des mécanismes de défense que nous avons recensés, aucun n'est satisfaisant en tout point.

Nous montrons dans le chapitre 6 qu'une partie des nœuds qui composent le réseau P2P d'Ethereum montrent certaines caractéristiques pouvant s'apparenter à des nœuds Sybil. Pour y remédier, nous proposons dans le chapitre 7 un système de supervision distribuée de ces potentiels nœuds Sybil ainsi qu'un mécanisme de révocation locale de ces dits nœuds.

Les prochains chapitres présentent respectivement nos contributions dans l'analyse des réseaux P2P de Bitcoin et d'Ethereum.

Deuxième partie

Supervision et analyse des réseaux P2P de Bitcoin et Ethereum

Chapitre 3

Caractéristiques du réseau P2P non structuré de Bitcoin

Sommaire

Introduction

L'état de l'art sur la supervision des réseaux pair-à-pair des deux systèmes de blockchain les plus populaires, à savoir Bitcoin et Ethereum, est lacunaire en ce qui concerne l'étude de la fiabilité du réseau dans son ensemble. Un autre point sur lequel les précédentes études sont lacunaires est la reproductibilité des expériences : les outils de supervision ne sont, pour la plupart, pas open-source et les jeux de données ne sont pas publics. En outre, la supervision rigoureuse d'un réseau P2P est complexe à réaliser, notamment à cause de leur aspect entièrement décentralisé, de la complexité intrinsèque des couches protocolaires mises en œuvre pour l'établissement dudit réseau et de leur documentation parcellaire. Dans cette partie, nous proposons deux architectures de supervision pour Bitcoin et Ethereum. La première se base sur un outil d'exploration des pairs

Bitcoin existant, pour lequel nous avons ajouté quelques fonctionnalités. Nous avons en revanche entièrement développé les outils de la seconde architecture concernant Ethereum. L'implantation de ces architectures est entièrement open-source (code du crawler et scripts d'analyse) et les jeux de données que nous avons constitués sont accessibles publiquement.

Dans ce chapitre, notre objectif est d'étudier les caractéristiques réseau et pouvant affecter la fiabilité du réseau P2P de Bitcoin. Nous étudierons le réseau P2P Ethereum dans le prochain chapitre.

3.1 Stratégie de supervision

3.1.1 Primitives de découverte des nœuds du réseau P2P Bitcoin

Concernant Bitcoin, nous nous concentrons sur l'implémentation du client Bitcoin Core [Thed], qui est l'implémentation officielle, documentée par la communauté Bitcoin [Thec]. Les autres implémentations non officielles (btcd [Theh], bitcoinj [Theg] ou bcoin [Thea] pour citer quelques autres implémentations populaires) sont très similaires, mais nous ne discuterons pas de leurs caractéristiques spécifiques ici.
Il existe quatre principaux types de messages dans le réseau P2P de Bitcoin :

- les messages de découverte du réseau ;
- les annonces de transactions ou de blocs ;
- les demandes de transactions ou de blocs ;
- l'envoi des transactions ou blocs.

Nous nous concentrerons sur les messages de découverte du réseau P2P [DW13] [FSW14] [HKZG15] [GBE+18] [DBG18].

Pour rappel, il existe deux types de nœuds dans le réseau P2P Bitcoin : les nœuds complets (*full node*) et les nœuds légers (*light node*). Un nœud complet télécharge chaque bloc et chaque transaction et vérifie qu'ils respectent toutes les règles de consensus de Bitcoin. Si ce n'est pas le cas, les blocs ou les transactions invalides sont rejetés, quel que soit l'état global du réseau à leur sujet. En revanche, un nœud léger télécharge uniquement les en-têtes des blocs pour vérifier l'authenticité des transactions. Cette méthode est appelée vérification simplifiée des paiements (SPV : *Simplified Payment Verification* en anglais). Ce type de nœud s'appuie sur des nœuds complets, qui deviennent des tiers de confiance, pour récupérer les données complémentaires pour valider et mettre à jour son état selon les besoins de l'utilisateur. Les nœuds légers et complets utilisent des clients logiciels différents, il existe des clients complets (*full clients*) et légers (*light clients*). Pour utiliser Bitcoin de manière privée et sans avoir besoin de faire confiance à un tiers, un utilisateur doit utiliser un portefeuille (*wallet*) basé sur un nœud complet.

Un nœud Bitcoin complet, lorsqu'il est démarré, doit se connecter à d'autres pairs actifs du réseau pour synchroniser l'état de sa blockchain locale avec celui convenu par le réseau. Au tout premier démarrage, un nœud ne connaît pas d'autres pairs du réseau. Classiquement, il utilisera quelques serveurs DNS (aussi appelé *DNS seeders* en anglais) pré-configurés dans le client auxquels il demandera les informations de quelques nœuds qui acceptent des connexions. En fait, les serveurs DNS parcourent continuellement le réseau pour créer une liste de pairs actifs et joignables. Lorsqu'ils reçoivent une requête DNS d'un nouveau nœud, ils répondent avec un sous-ensemble d'adresses IP sélectionnées qu'ils connaissent. Le nombre d'adresses IP à retourner n'est pas spécifié formellement, chaque implantation de *DNS seeder* peut être différente mais dans la pratique on observe que ce nombre est autour de 25. La liste des noms d'hôtes DNS à interroger est pré-configuré dans le client Bitcoin (quelques exemples : seed.bitcoin.sipa.be, dnsseed.bluematt.me, dnsseed.bitcoin.dashjr.org).

Un nœud dispose de deux bases de données dans lesquelles il stocke les adresses IP des autres pairs du réseau : la table *new* et la table *tried*. Ces deux tables sont organisées en buckets, 1024 buckets pour la première et 256 pour la seconde. Chaque bucket peut contenir un maximum de 64 entrées, ce qui porte la taille théorique maximale de la table *new* à 65 536 nœuds et de *tried* à 16 384. La table `new` contient les adresses des pairs auxquels le nœud n'a pas encore essayé de se connecter. La table `tried` contient les adresses des pairs auxquels le nœud a pu se connecter avec succès et qui sont accessibles.

Une fois que la base de données locale a été alimentée avec quelques pairs actifs, le nœud essaie de se connecter à 8 pairs par défaut en utilisant des connexions TCP non chiffrées (appelées connexions sortantes). En outre, le nœud peut être configuré (redirection de port, accès autorisé dans le pare-feu) pour accepter des connexions (appelées connexions entrantes) d'autres pairs, jusqu'à 117 par défaut. Un tel nœud — appelé *listening node* (nœud d'écoute) — peut émettre des messages `ADDR` non sollicités pour annoncer aux nœuds auxquels il est connecté (que l'on peut aussi appeler « voisins ») qu'il accepte les connexions entrantes. Ces voisins peuvent relayer ce message à leurs propres voisins et la propagation peut se poursuivre de cette manière en suivant un protocole d'inondation (*flooding*). Un nœud peut également demander explicitement à découvrir d'autres pairs actifs dans le réseau en envoyant un message de type `GETADDR` à ses voisins. Pour déterminer le nombre d'adresses à inclure dans une réponse, il calcule m le minimum entre 23 % du nombre de nœuds contenus dans tous les buckets et 2 500. Ensuite, il répond par des messages `ADDR` contenant les informations (adresse IP, n° port, horodatage) des m pairs récupérés aléatoirement dans les buckets (un seul message `ADDR` peut contenir jusqu'à 1000 entrées de buckets). Au maximum, un nœud enverra 3 messages contenants chacun respectivement 1000, 1000 et 500 adresses issues de ses tables.

C'est ainsi que les messages `ADDR` sont documentés pour le client Bitcoin Core, mais nous avons remarqué que ce n'est plus le cas pour les versions publiées après 2014($>$ v0.10.0). Depuis lors, un nœud envoie au maximum 1000 adresses dans un message. La liste des adresses qui contient 23 % du nombre de nœuds est rognée à 1000. La raison n'est pas documentée et n'est pas claire, même au sein de la communauté et les développeurs prévoient de supprimer cette limitation à l'avenir [Theb].

En outre, un nœud maintient ses connexions à jour en vérifiant périodiquement l'état des nœuds auxquels il est connecté en émettant des messages `PING` au niveau applicatif (en TCP, pas ICMP) et en attendant les réponses `PONG` (TCP). Les deux types de connexion (entrante et sortante) sont utilisés pour diffuser les messages contenant les transactions et les blocs (annonce, demande et envoi).

3.1.2 Méthodologie de supervision

Du 7 février au 2 mars 2020, nous avons exécuté un nœud de supervision sur le réseau principal de Bitcoin. Nous avons forké (au sens de Github fork) et amélioré le crawler open source utilisé par le site bitnodes.io et nous avons également publié notre version sur Github [Eisb] [14]. Nous avons étendu ce projet afin d'ajouter quelques fonctionnalités manquantes :

— le suivi du taux de découverte des nœuds pour valider la méthodologie ;
— la récolte et l'exportation de tout le contenu des buckets des nœuds ;
— l'ajout de toutes les fonctions permettant de calculer les statistiques et de générer les graphiques que nous présentons dans ce chapitre.

La méthodologie de ce logiciel est résumée dans le schéma 3.1. Cette méthodologie implique

14. `https://github.com/jpeisenbarth/bitnodes/tree/add_statistics`

d'abord un crawler qui tente d'établir simultanément des connexions avec les pairs du réseau Bitcoin et ensuite, un autre programme qui fonctionne en parallèle et qui vérifie continuellement la connectivité des nœuds découverts. Ce second programme se connecte à tous les nœuds trouvés par le crawler et leur envoie périodiquement (toutes les 60 secondes) un message `PING` (TCP) afin de vérifier s'ils sont toujours connectés. Lorsqu'un crawl est terminé, les nœuds marqués comme accessibles par ce programme sont exportés avec des informations supplémentaires telles que la géolocalisation, le nom d'hôte, le client Bitcoin utilisé.

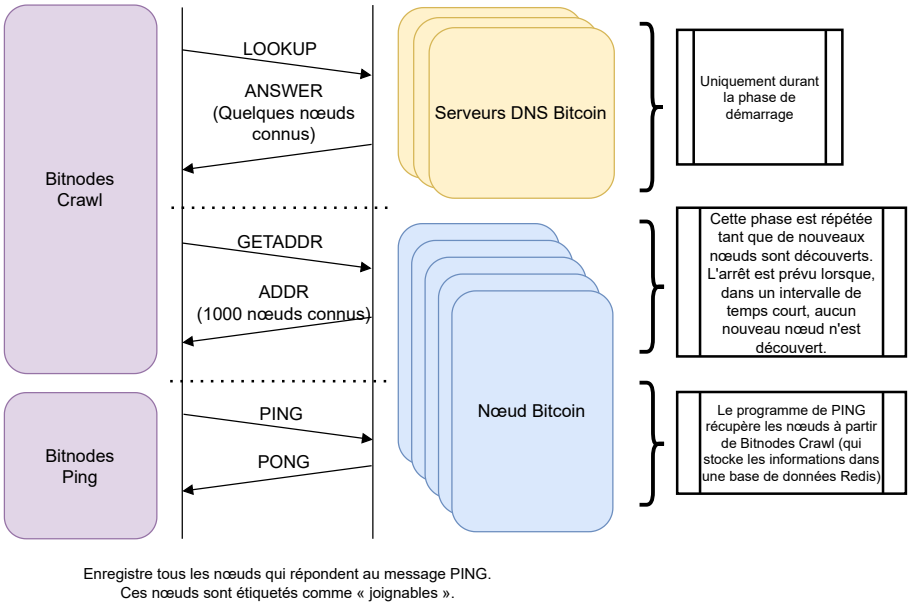

FIGURE 3.1 – Stratégie d'exploration du réseau P2P repris du programme Bitnodes

On notera que dans l'ensemble de données que nous avons rendu public, les adresses IP sont pseudo-anonymisées conformément aux recommandations de l'ICANN [Int18] qui offre un compromis acceptable entre utilité et confidentialité. Ceci nous a en outre permis de faire valider la diffusion du jeu de données par le COERLE de l'INRIA. Plus précisément, un hachage salé est fourni et le dernier octet de l'adresse est mis à 0.

De plus, les nœuds qui exécutent une version du protocole Bitcoin antérieure à 70001 n'ont pas été pris en compte par le crawler en raison de problèmes de compatibilité avec les versions antérieures du protocole. Cela correspond au client Bitcoin Core v0.8.0 et inférieur (avant février 2013). Pappalardo et al. [PDMCA18] ont montré que, déjà en 2016, la grande majorité des nœuds du réseau (plus de 80%) exécutaient une version du protocole plus récente que 70001. Par conséquent, nous considérons que le crawler pouvait raisonnablement ignorer les nœuds qui exécutent une version plus ancienne du protocole.

3.1.3 Configuration du serveur

La machine que nous avons utilisée est un ordinateur sous Ubuntu 18.04 LTS avec un processeur Intel Xeon E5-2420 v2 6 16 Go de RAM et une liaison réseau de 1 Gb/s qui établit des connexions avec tous les pairs IPv4 joignables — c'est-à-dire les nœuds ouverts aux connexions entrantes — du réseau P2P de Bitcoin. Pour gérer le grand nombre de connexions réseau nécessaires, nous avons dû augmenter la limite de fichiers ouverts de notre système GNU/Linux

à 1.000.000. Nous avons également défini certains paramètres réseau du noyau, comme spécifié ci-après :

```
net.core.rmem_default=33554432
net.core.wmem_default=33554432
net.core.rmem_max=33554432
net.core.wmem_max=33554432
net.ipv4.tcp_rmem=10240 87380 33554432
net.ipv4.tcp_wmem=10240 87380 33554432
net.ipv4.ip_local_port_range=2000 65500
```

La machine a fait fonctionner le nœud de crawling 24 heures sur 24 et 7 jours sur 7 pendant toute la période, qui, pour rappel, s'étend du 7 février au 2 mars 2020. Il est important de noter que notre infrastructure ne dispose pas d'un lien IPv6 pour tester et intégrer ces nœuds. Ainsi, tous les nœuds découverts sont des nœuds IPv4.

En 2019 et début 2020, ils représentaient environ 75% du réseau alors que les nœuds IPv6 et Tor représentaient 10% et 15% du réseau respectivement. En observant aujourd'hui les statistiques sur le site `bitnodes.io`, on remarque que la part des nœuds IPv4 a considérablement diminué, au profit de nœuds TOR. Par exemple, en mai 2022, 38 % des nœuds fonctionnent en IPv4, 55 % des nœuds utilisent Tor, et 7 % utilisent IPv6. À l'époque où nous menions ces travaux sur Bitcoin, il était raisonnable de considérer que la tendance globale du réseau IPv4 pouvait être extrapolée à l'ensemble du réseau, mais cela n'est dorénavant plus le cas même si d'autres études récentes semblent montrer que les propriétés du réseau sont restées similaires.

3.1.4 Jeux de données collectés

Nous avons ainsi récolté un jeu de données[15], dont un extrait est présenté en annexe A. Il se présente sous la forme de deux dossiers, « crawl » et « export ».

Les fichiers du répertoire « crawl » contiennent des informations et des statistiques sur la découverte des pairs du réseau. Les fichiers JSON de ce dossier, dont le format du nom est « date_du_crawl.json » contiennent les informations (adresse IPv4, port TCP, un champ de bits représentant les fonctionnalités offertes par le nœud, et la hauteur de la blockchain possédée) des pairs joignables découverts par Bitnodes. L'autre type de fichier JSON (mais encodé en utilisant le format Pickle de Python), dont le format du nom est « nodes_per_getADDR_date.pickle.bz2 » contient des informations sur la popularité des pairs dans les tables de routage. Les deux autres types de fichiers, au format CSV, dont les noms sont « nodes_per_getADDR_date.csv » et « up_nodes_per_seconds_date.csv » permettent de générer les statistiques sur le nombre de nœuds du réseau et sur la validation de la couverture de la supervision. Ces statistiques sont présentées plus loin dans ce chapitre.

Le répertoire « export » quant à lui contient les fichiers JSON des résultats de tous les crawls effectués durant la période de février à mars 2020. Ces fichiers contiennent les mêmes informations sur les pairs joignables que les fichiers JSON du répertoire « crawl », mais celles-ci sont agrémentées de quelques informations supplémentaires, à savoir : la version du protocole utilisé par le nœud, la version du client, sa date de dernière connexion, son nom d'hôte, ainsi que des informations de géolocalisation de l'IP (ville, code pays, latitude, longitude, fuseau horaire, numéro d'AS, nom de l'AS).

15. Disponible ici `https://concordia-btc-p2p.lhs.loria.fr/`

3.1.5 Validation du crawler

Afin de valider la qualité et la cohérence de nos mesures, nous voulons évaluer qu'un crawl se termine lorsque tous les pairs possibles sont découverts. Un crawl se déroule comme suit :

1. le nœud d'exploration est démarré en recevant un sous-ensemble de nœuds joignables (généralement 25) de la part de des DNS seeders connus dont les adresses [16]sont préconfigurées dans le client ;
2. il essaie d'établir une connexion avec les nœuds nouvellement découverts ;
3. il envoie ensuite des messages `GETADDR` à tous les nœuds auxquels il est connecté ;
4. le crawler se déconnecte d'un nœud une fois qu'il a reçu les messages `ADDR` en réponse à ses `GETADDR` ;
5. il reboucle à l'étape 2 jusqu'à ce qu'il n'y ait plus de nouveaux nœuds découverts pendant un certain temps (nous avons configuré ce temps à 10 secondes pour notre instance du crawler).

Plus précisément, les nœuds nouvellement découverts sont stockés dans une liste temporaire en attendant que le nœud d'exploration tente d'établir une connexion avec eux. Ces nœuds sont récupérés à partir des réponses aux messages `GETADDR` envoyées par les nœuds auxquels le crawler est connecté. Cette liste d'adresses est filtrée afin d'écarter les nœuds périmés (vus en ligne pour la dernière fois plus de 8 heures avant l'heure actuelle de l'exploration) pour des raisons d'efficacité, car les nœuds plus anciens sont plus susceptibles d'être déconnectés. Si ceux-ci sont toujours présents, ils seront découverts via d'autres pairs qui les auront contactés plus récemment. Une exploration s'arrête lorsqu'il n'y a plus de nœuds en attente d'être contactés pendant 10 secondes consécutives. En pratique, un crawl ne dépasse pas 3 minutes et le programme est configuré pour redémarrer un nouveau crawl toutes les 4 minutes. Comme le montre le graphique 3.2, un crawl cesse d'apprendre de nouveaux nœuds après 2 à 3 minutes et le nombre de nouveaux nœuds découverts au fil du temps diminue après un pic au démarrage.

Nous avons également déployé cinq nœuds témoins Bitcoin Core indépendants, dont la version du client est v0.20.1. Nous n'avons pas modifié le code source, ils ont exécuté le client Bitcoin Core officiel récupéré à partir de Github. Ils ont rejoint le réseau P2P et quatre d'entre eux avaient environ 30 connexions actives tandis que le cinquième nœud avait 70 connexions actives. Le crawler a pu retrouver tous les nœuds témoins lors de l'exploration du réseau P2P. La méthodologie du robot d'exploration peut raisonnablement être considérée comme efficace pour trouver tous les nœuds accessibles du réseau P2P Bitcoin.

3.2 Métrologie réseau et fiabilité du réseau Bitcoin

Dans cette section, nous analysons plusieurs mesures qui sont dérivées des crawls du réseau Bitcoin. Habituellement, une cyberattaque comporte une phase de reconnaissance au cours de laquelle l'attaquant recueille un grand nombre d'informations sur la cible qui peuvent être exploitées pour trouver une ou plusieurs faiblesses. Par exemple, une attaque par partitionnement du réseau nécessite que l'attaquant mette en évidence les nœuds d'intérêt qui doivent être déconnectés du réseau. Pour chaque aspect du réseau, nous considérerons ce qu'il indique concernant la robustesse du réseau contre les attaques.

Toutes ces mesures peuvent servir de faisceau d'indices pour déceler une potentielle attaque globale du réseau en direct.

16. dnsseed.bitcoin.dashjr.org, dnsseed.bluematt.me, seed.bitcoin.sipa.be, seed.bitcoinstats.com, seed.bitcoin.sprovoost.nl et seed.bitnodes.io

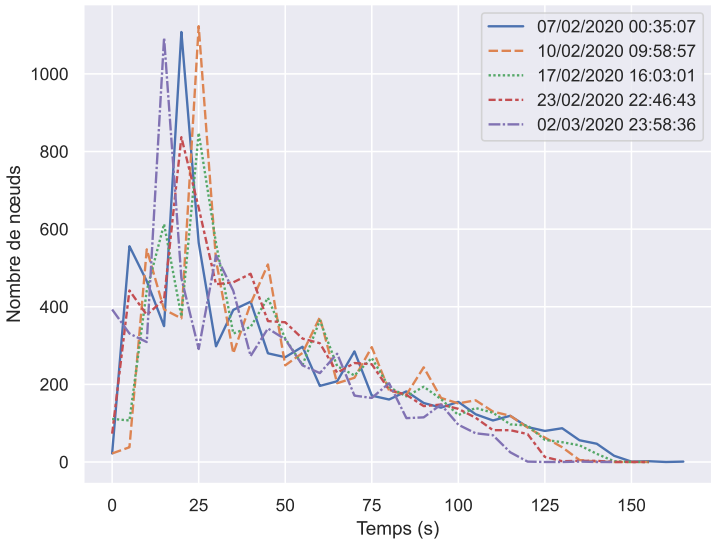

FIGURE 3.2 – Nombre de nouveaux nœuds joignables découverts tout au long d'un crawl (5 crawls différents sont représentés)

3.2.1 Nombre de nœuds

Comme le montre le tableau 3.1, le crawler a pu découvrir en moyenne 7 218 de nœuds uniques dans le réseau P2P au cours d'un seul crawl. Le nombre minimum de nœuds accessibles trouvés était de 6 856 [17] et le maximum 8 103.

TABLE 3.1 – Nombre total de nœuds (accessible et non accessibles) découverts par le crawler

Nombre (par crawl)	Minimum	Maximum	Moyenne	Médiane
Nœuds découverts	5 506	7 623	7 218	7 310
Nœuds accessibles	6 856	8 103	7 783	7 870
Nœuds uniques dans les réponses aux messages `GETADDR`	19 164	35 076	25 888	25 895

Le programme chargé de maintenir la connexion ouverte avec tous les nœuds découverts était connecté en moyenne à 7 783 nœuds tout au long de la période. La différence entre les nœuds découverts et les nœuds atteignables peut s'expliquer par le fait qu'une petite partie des nœuds peut être manquée par un crawl particulier mais trouvée par les autres et le second programme qui garde la trace des nœuds accessibles est régulièrement mis à jour avec les nouveaux nœuds découverts par le crawler. Le réseau P2P était composé de 6 856 nœuds accessibles au minimum et 8 103 au maximum.

17. Pour calculer le minimum, nous avons omis les données du 21 février parce que nous avons dû redémarrer le programme et il lui a fallu plusieurs crawls pour se connecter au nombre exact de nœuds atteignables du réseau P2P. Si nous avions gardé ces données, le minimum, le maximum, la moyenne et la médiane auraient été de 6 393, 8 103 (inchangé), 7 785, et 7 869, respectivement

Nous avons également une image globale du nombre total de nœuds référencés dans le réseau P2P (pas seulement ceux qui sont accessibles). Comme d'habitude, le crawler envoie un seul message `GETADDR` à un nœud joignable nouvellement découvert, qui répond avec 1000 nœuds contenus dans tous ses buckets (voir 3.1.1). Nous considérons ici les nœuds uniques découverts par crawl sans tenir compte de leur accessibilité, et nous avons constaté qu'il y a 200 000 nœuds uniques en moyenne 200 000 nœuds uniques récupérés à partir des buckets de tous les nœuds accessibles. En raison du chevauchement possible au niveau des réponses, nous pouvons déduire que ce nombre, bien qu'approximatif, doit être représentatif de la taille moyenne de l'ensemble du réseau P2P (nœuds publics et non publics).

3.2.2 Distribution des différents clients

Les nœuds joignables que nous découvrons sont tous des nœuds publics (c'est-à-dire que l'utilisateur qui a déployé un nœud a correctement configuré le pare-feu et/ou la redirection de port si nécessaire) mais nous ne pouvons pas conclure s'il s'agit de nœuds complets ou légers avec la seule information qu'ils sont publics.

Pour répondre à cette question, nous avons classé les nœuds publics découverts en fonction du client (également appelé *user agent*) qu'ils utilisaient. Pour un nœud complet, le client officiel est `Bitcoin Core` (son user agent est`Satoshi`). C'est le client le plus largement déployé que nous ayons trouvé, avec jusqu'à 98,36 % des nœuds du réseau qui exécutent ce client. Des dizaines de versions de ce client sont utilisées. Pendant toute la période de crawl, ces trois versions sont les plus populaires :

— `0.18.0` (mai 2019) ;
— `0.18.1` (août 2019) ;
— `0.19.0.1` (novembre 2019).

Leur ordre relatif varie au cours de la période mais ils représentent toujours environ 65 % du réseau exploré.

D'autres clients sont présents dans l'ensemble de données, mais dans un très faible pourcentage (moins de 1 %). Leur user-agent sont : `bcoin`, `btcd` et `bitcoin_unlimited`. Nous avons également trouvé un nombre encore moins significatif d'autres clients que nous ne pouvions pas classer car nous n'avons pas pu trouver d'informations fiables à leur sujet. Leur user-agent sont par exemple : `Aurum`, `CKCoind`, `therealbitcoin.org`, `BitflateCore`, `MultiChain`.

Nous pouvons en déduire qu'une très grande majorité (environ 99 %) des nœuds publics découverts sont des nœuds complets et cette majorité exécute le client officiel de Bitcoin.

3.2.3 Distribution des versions du client officiel et fiabilité

Pour assurer la fiabilité globale, les nœuds d'un réseau P2P doivent utiliser la dernière version des clients afin de bénéficier des derniers correctifs de sécurité. Pour évaluer cette information, nous classons les clients en fonction de leur date de sortie officielle [Thef] afin d'observer comment les utilisateurs maintiennent le client à jour. Comme le montre la figure 3.3, une majorité de nœuds est à jour en exécutant une version du client dont la date de sortie est comprise dans l'année précédant la date de sortie de la version courante. Pour que le graphique reste lisible, nous n'avons pas affiché la distribution temporelle des versions publiées avant 2016 : cela ne concerne qu'une centaine de nœuds dont la version du client a été publiée entre 2013 et 2015 comme on peut le voir dans le graphique 3.4.

Nous pouvons voir sur le graphique 3.3 qu'une quantité significative de clients (plus de 30 %) exécute des versions qui pourraient être considérées comme dépréciées (plus d'un an), ce qui

TABLE 3.2 – Quelques CVE importantes qui affectent le client Bitcoin Core que l'on retrouve dans une partie des nœuds déployé en mars 2020

	Versions affectées	Sévérité [NIS] (sur 10 points)	Proportion du réseau vulnérable
CVE-2016-10724 (1)	$<$ 0.13.0	7,5	1,62%
CVE-2016-10725 (2)	$<$ 0.13.0	7,5	1,62%
CVE-2017-18350 (3)	$<$ 0.15.1	5,9	9,42%
CVE-2018-17144 (4)	0.14.0 to 0.16.3	7,5	7,25%
CVE-2018-20586 (5)	$<$ 0.17.1	5,3	22,30%
CVE-2018-20587 (6)	0.12.0 to 0.17.1	5,5	21,96%

peut avoir des conséquences sur la fiabilité lorsque certaines vulnérabilités sont exploitées.

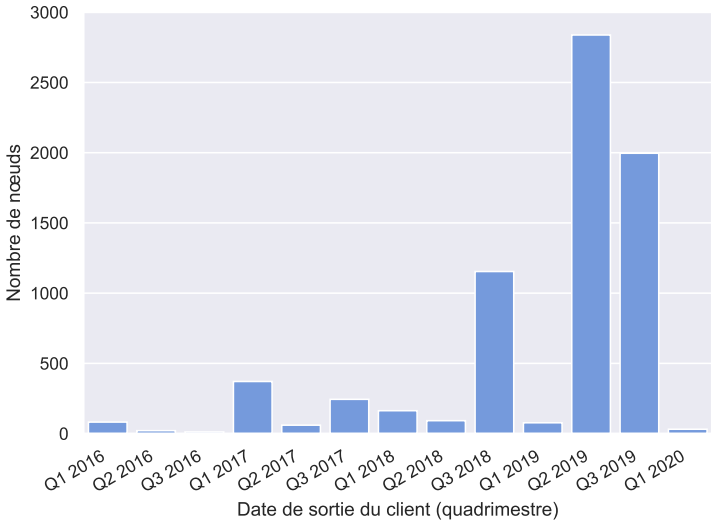

FIGURE 3.3 – Distribution temporelle des versions du client `Bitcoin Core`, le 2 mars 2020. Plusieurs versions consécutives peuvent avoir été publiées durant le même quadrimestre.

Nous avons donc analysé certaines vulnérabilités et expositions communes (CVE) qui affectent le client Bitcoin Core et qui sont répertoriées dans le tableau 3.2. Nous voulions comprendre comment un client à jour pourrait contribuer à réduire les risques d'une attaque globale au niveau du réseau.

La CVE (5) et la CVE (6) peuvent être exploitées via le même composant : l'implémentation RPC. Dans le premier cas, cela permettrait à un attaquant d'injecter des données arbitraires dans le journal de débogage et dans le second cas, de voler des devises. L'impact de cette dernière vulnérabilité est plus élevé et critique mais très peu probable puisqu'elle nécessite de surcroît un accès local au nœud. Ces deux vulnérabilités concernent une partie importante du réseau (plus de 20 %) et les utilisateurs courent le risque d'être confrontés à ces deux attaques en ne mettant

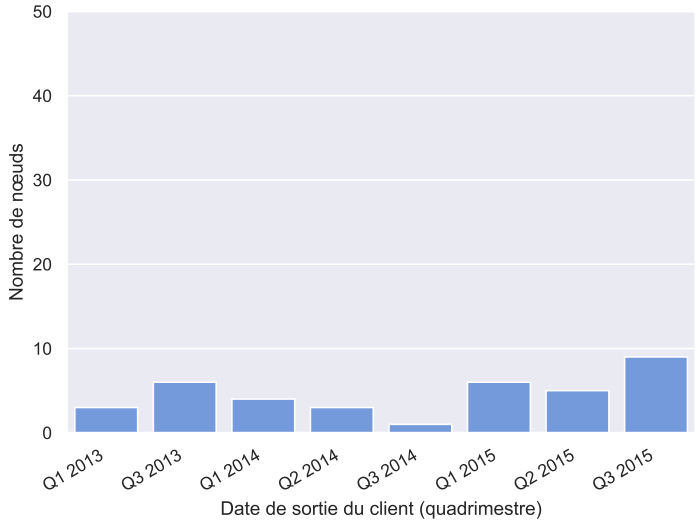

FIGURE 3.4 – Distribution temporelle des versions (datant d'avant 2016) du client `Bitcoin Core`, le 2 mars 2020.

pas à jour leurs nœuds.

Les CVE (1) et CVE (2) concernent également le même composant : le système d'alerte. Il s'agit d'un système introduit dans la version 0.3.10 et retiré dans la version 0.13.0, qui permettait aux détenteurs d'une clé privée associée au système d'alerte de diffuser des messages signés à tous les clients du réseau pour les informer de certains événements importants (tels que des forks accidentels). Dans ce système d'alerte, il y a un message spécial (une « super » alerte) qui outrepasse tous les autres messages d'alerte. Cela devait par exemple permettre d'annoncer que la clé privée utilisée pour les messages d'alerte « normaux » était compromise et que le client `Bitcoin Core` devait être mis à jour. La CVE (2) permet à un attaquant de bloquer la réception d'une « super » alerte avec un message d'alerte classique alors que la CVE (1) permet à un attaquant de réaliser un déni de service (par épuisement de la mémoire).

Il y a deux autres CVE qui permettent un déni de service : CVE (3) et (4). La première concerne la version de Bitcoin Core antérieure à la version 0.15.1 qui connaîtrait un débordement de tampon basé sur la pile si le client utilise un proxy SOCKS contrôlé par l'attaquant et le second concerne les versions de 0.14.0 à 0.16.3 qui subiraient un crash de l'application en raison de la duplication des entrées envoyées par les mineurs.

Nous pensons que les CVE qui permettent le déni de service à travers Internet sont les plus préoccupantes pour le réseau. Elles concernent environ 10% du réseau, mais elles pourraient être exploitées par une attaque globale dont le but est d'abaisser la majorité du réseau, comme décrit dans les travaux [HKZG15] et [NKMS16]. Pour que le réseau dans son ensemble soit sécurisé, cela implique que tous les utilisateurs doivent maintenir leur client à jour. Le fait que les clients sont relativement à jour montre que cette considération n'est pas particulièrement inquiétante pour le réseau Bitcoin, à la période étudiée.

En outre, une propriété intéressante du réseau à analyser est la réactivité des pairs à mettre à jour leur client lorsqu'une nouvelle version est publiée. Dans le graphique 3.5, nous montrons la distribution temporelle des clients quelques jours après la sortie de la version `0.20.0` (début juin 2020). Nous pouvons constater qu'il y a un nombre significatif de premiers utilisateurs qui ont mis à jour leurs clients rapidement. En outre, depuis mars (voir le graphique 3.3), il y a

un nombre important de clients qui ont mis à niveau leurs clients vers une version plus récente (c'est-à-dire qu'il y a un passage de Q1 2019 à Q1 2020). Cependant, nous pouvons encore remarquer qu'un nombre important d'utilisateurs continue à utiliser des versions dépréciées du client publiées avant le deuxième trimestre 2019. Pour les versions antérieures à 2016, la tendance est équivalente à celle du graphique 3.3. Un petit effort de la communauté pour inciter à la mise à niveau des versions dépréciées pourrait être bénéfique pour la fiabilité du réseau dans son ensemble, même si ce n'est pas actuellement inquiétant comme nous l'avons dit précédemment.

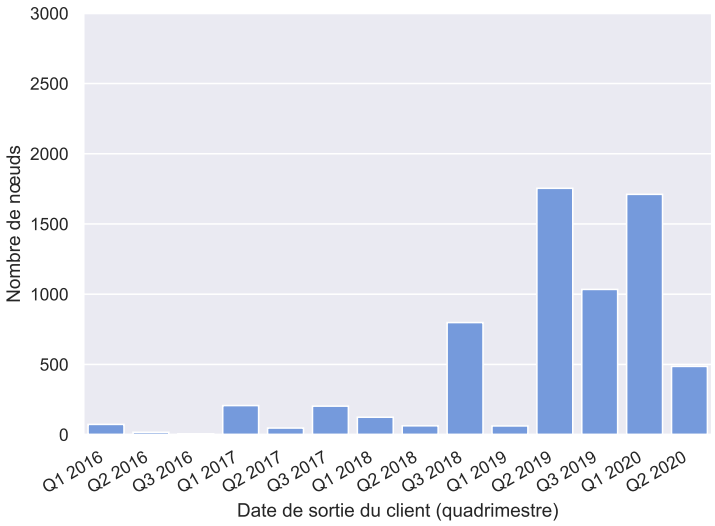

FIGURE 3.5 – Distribution temporelle des versions du client `Bitcoin Core`, le 7 juin 2020.

3.2.4 Distribution géographique et entre ASes

Nous savons maintenant que les nœuds publics sont des nœuds complets, et nous pouvons nous demander si leur répartition géographique reste stable au cours d'une journée.

Nous montrons dans le graphique 3.6 le nombre de nœuds joignables le 15 février dans le monde divisé en trois zones : Europe/Afrique/Moyen Orient, Amérique du Nord/Sud et Asie/Océanie. Nous avons utilisé les fuseaux horaires comme critère pour regrouper les zones. Comme nous pouvons le constater, le nombre de nœuds ascendants au fil du temps est assez constant dans les trois zones. Les nœuds publics complets ne sont pas éteints à la tombée de la nuit. Ils fonctionnent très probablement sur des serveurs plutôt que sur les ordinateurs personnels des utilisateurs, car la rotation quotidienne périodique par zone géographique inhérente aux réseaux P2P de partage de fichiers n'est pas observée ici. Nous validerons cette hypothèse en montrant que les nœuds sont majoritairement hébergés chez des fournisseurs cloud.

Nous pouvons également remarquer que, contrairement à la distribution géographique des nœuds de minage, il y a plus de nœuds publics complets déployés sur le continent européen ou en Amérique du Nord qu'en Asie. Par exemple, dans l'ensemble de notre jeu de données, à tout moment, il y a en moyenne environ 3900 nœuds hébergés en Europe, 2150 en Amérique du Nord et 1200 en Asie. L'absence de tendance quotidienne ainsi qu'une répartition géographique équilibrée sont de bonnes propriétés pour les systèmes distribués, les rendant plus robustes aux pannes.

Généralement, une géolocalisation équilibrée des nœuds est une bonne propriété du réseau,

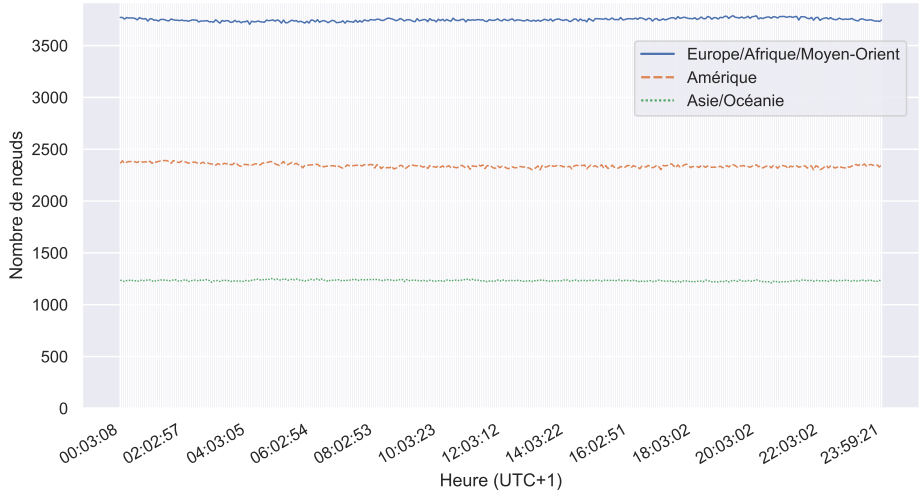

FIGURE 3.6 – Distribution géographique des nœuds publics sur une journée

puisque son but est de propager rapidement les transactions et les blocs à travers tout le réseau : une distribution géographique équilibrée est plus résiliente aux problèmes localisés. Mais pour mieux évaluer la concentration des nœuds, nous avons également analysé leur distribution au niveau du réseau dans les systèmes autonomes (*Autonomous systems* en anglais, abrégé ASes). En février 2020, 35% des pairs étaient hébergés dans 10 AS sur 1566 totaux (0.6%). Huit de ces dix AS sont des fournisseurs de cloud. Cette statistique dénote une bonne décentralisation des nœuds, ce qui est une propriété très souhaitable pour la robustesse du réseau.

3.2.5 Taux d'attrition (*churn*)

Nous voulons ensuite savoir si le taux d'attrition est faible et stable. Nous avons surveillé les déconnexions et les reconnexions des nœuds pendant toute la période de mesure. Cela signifie que nous avons observé les nœuds qui quittaient ou rejoignaient le réseau d'un jour à l'autre. Comme le montre le graphique 3.7, le taux de nœuds quittant ou rejoignant le réseau est assez stable entre 5% et 6% par jour. Concernant la courbe en pointillés, nous n'avons pas distingué les nouveaux nœuds qui rejoignaient le réseau et ceux qui rejoignaient le réseau après l'avoir quitté peu de temps auparavant. Nous pouvons voir qu'il y a un pic de déconnexions le 27 février. Nous avons décidé de surveiller le taux de désabonnement d'une heure à l'autre les 26 et 27 février. Comme le montre la Fig. 3.8, le pic s'est produit le 27 février entre 00:03 (UTC+1) et 13:03 (UTC+1), le nombre maximum de déconnexions se produisant entre 00:03 (UTC+1) et 01:03 (UTC+1). Dans notre ensemble de données, nous n'avons pas observé de taux de reconnexion par la suite qui aurait annulé ce taux de déconnexion.

Malgré ce pic, nous pouvons observé que le réseau de nœuds joignables est stable même si presque tous les nœuds subissent une déconnexion à un moment donné, comme le montrent Imtiaz et al. [ISTY19]. On notera qu'on ne peut pas distinguer un nœud raté par le crawler d'une déconnexion mais l'étude de la couverture et la validation du crawler ainsi que l'agrégation des crawls sur une journée doit limiter fortement ce phénomène.

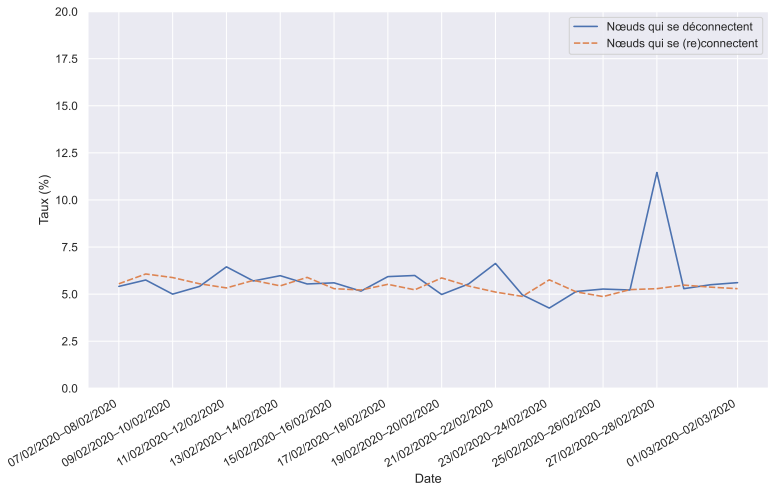

FIGURE 3.7 – Pourcentage de nœuds qui quittent ou rejoignent le réseau, sur un jour

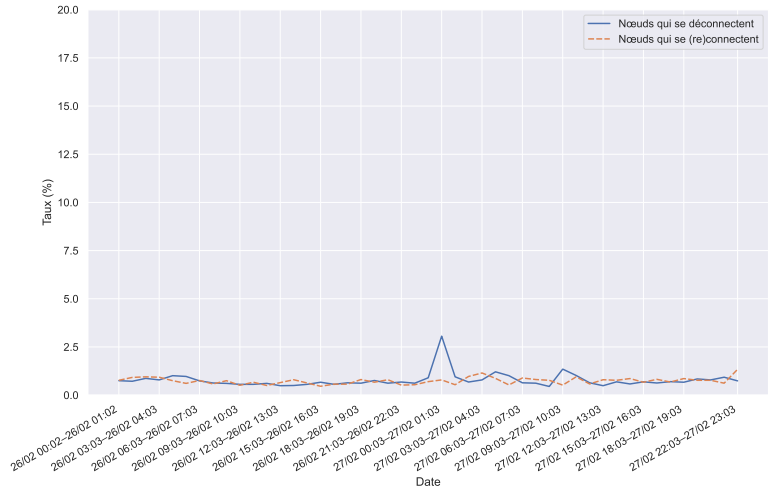

FIGURE 3.8 – 26 et 27 février : pourcentage de nœuds qui quittent ou rejoignent le réseau, sur une heure

3.2.6 Distribution de la popularité des pairs dans la liste des contacts

Nous nous sommes ensuite intéressés au contenu des réponses de ces messages `GETADDR`. En particulier, nous voulions savoir à quelle fréquence une adresse IP donnée apparaît dans les buckets des nœuds joignables. Comme précédemment, nous avons agrégé une heure de crawls pour avoir des résultats plus fiables. Nous pouvons voir dans le graphique 3.9 les occurrences des adresses IP, triées par ordre croissant. Il est intéressant d'observer la tendance de la courbe : une grande majorité des adresses IP n'apparaît que dans les buckets de quelques nœuds atteignables et quelques adresses IP ont tendance à apparaître dans les buckets d'un grand nombre de nœuds atteignables. Il est important de noter que toutes ces adresses IP ne sont pas nécessairement des nœuds joignables, elles peuvent aussi être des nœuds hors ligne (à court terme) ou des nœuds inaccessibles (derrière un NAT, des portefeuilles Bitcoin, etc.).

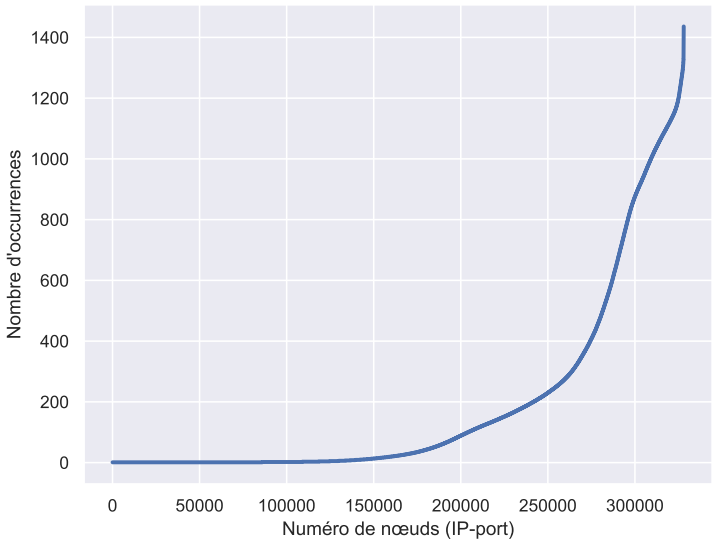

FIGURE 3.9 – Nombre d'occurrences de l'adresse IP d'un nœud dans les réponses `GETADDR`, collectées durant 1h de supervision

Nous avons observé que presque toutes les 150 000 premières adresses IP (les moins fréquentes) sont des nœuds inaccessibles (c'est-à-dire que notre crawler n'a pas réussi à établir une connexion). Alors que, dans les 1 000 adresses les plus fréquentes, environ 80 % sont des nœuds atteignables. Cette tendance est importante : il semble qu'une adresse IP qui est renvoyée dans une requête `GETADDR` par un nombre significatif de nœuds est plus susceptible d'être un nœud actif dans le réseau Bitcoin. Nous pouvons voir dans le graphique 3.10 les occurrences des nœuds atteignables uniquement. La courbe montre deux points d'inflexion clairs, ce qui permet de mettre en évidence 3 catégories principales de nœuds : les premiers 50 % qui sont les moins fréquents, les 25 % suivants montrent une popularité moyenne, mais non homogène et les derniers 25% sont très populaires.

Il serait intéressant de mesurer si les nœuds les plus populaires dans les tables de routage ont également un fort degré de connectivité au niveau de la topologie P2P.

Nous pouvons en déduire que le réseau a tendance à propager les adresses IP des nœuds joignables. Ceci est important pour la qualité du réseau P2P global. À l'inverse, la croissance exponentielle à la fin de la courbe montre que la popularité des nœuds est très déséquilibrée : seule une fraction de toutes les adresses IP récupérées est très populaire alors que les réseaux P2P sont plus résilients lorsqu'ils sont bien équilibrés. Ils constituent l'épine dorsale du réseau

et pourraient être des cibles de choix en cas d'attaque.

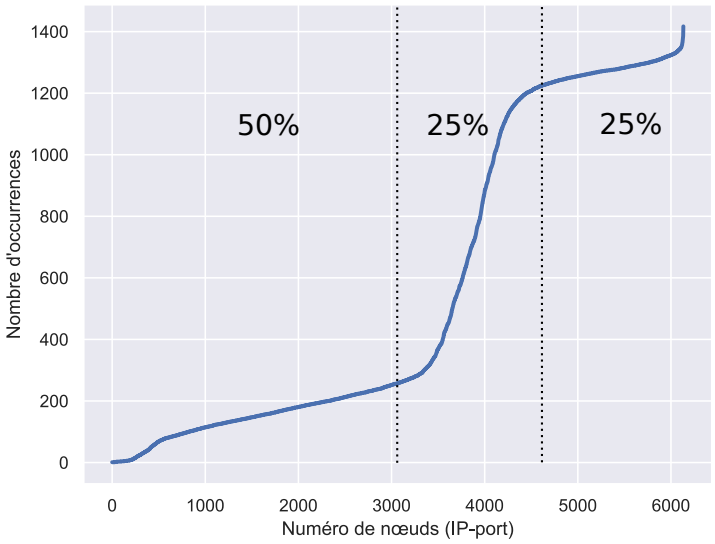

FIGURE 3.10 – Nombre d'occurrences de l'adresse IP d'un nœud public et accessible dans les réponses `GETADDR`, collectées durant 1h de supervision

3.2.7 Détection d'éventuelles attaques Sybil

Plusieurs études concernant les attaques Sybil sur le réseau P2P Bitcoin [BKP14, SMP19] montrent qu'un attaquant pourrait tirer parti de ce type d'attaque pour affaiblir le consensus (perturbation de la propagation des blocs/transactions, attaque (D)DoS, ...). Une technique simple pour détecter si une telle attaque est en cours est de vérifier s'il existe un nombre significatif de nœuds qui ont la même adresse IP ou qui fonctionnent dans le même sous-réseau. À notre connaissance, il n'y a pas de notion d'ID réseau dans Bitcoin (comme c'est le cas dans le réseau overlay d'Ethereum que l'on étudiera dans le chapitre 4). Notre analyse se limite à la recherche de nœuds sybils au niveau du couple adresse IP-n°port.

Dans notre ensemble de données, nous avons constaté qu'il n'y a que quatre sous-réseaux /24 dans lesquels 10 nœuds ou moins fonctionnent. De plus, le sous-réseau /24 dans lequel le plus grand nombre de nœuds fonctionne ne comporte que 19 nœuds. Compte tenu de la taille du réseau (environ 8000 nœuds), cela n'est pas significatif. Il semble qu'au cours de la période couverte par nos crawls, il n'y a pas de suspicion d'une attaque Sybil en cours. Afin d'évaluer la fiabilité du réseau, on pourrait surveiller l'ensemble du réseau en exécutant des crawls permanents et en utilisant cette technique pour détecter une éventuelle attaque Sybil.

3.3 Inférence de lien pour la découverte de la topologie réseau

3.3.1 Tentatives précédentes

La connaissance des liens réels entre les pairs est d'une importance capitale pour progresser dans notre compréhension du réseau P2P Bitcoin. Plusieurs travaux ont été réalisés sur l'inférence topologique, mais certains ont été rendus partiellement obsolètes par les contre-mesures introduites dans le code source du client Bitcoin pour éviter la découverte de la topologie [Thee, Theb].

En 2014, Biryukov et al. [BKP14] ont présenté une attaque Sybil contre le réseau P2P dont l'objectif est la désanonymisation des utilisateurs de Bitcoin. Ils ont exploité le fait que leur outil était connecté à tous les nœuds du réseau et pouvait injecter de fausses informations sur les pairs et suivre leur chemin et ainsi déduire la topologie. Cette méthodologie n'est pas adaptée à notre travail puisqu'il s'agit d'une attaque active qui perturbe fortement le réseau. Pour les mêmes raisons les travaux de Grundmann et al. [GNH18], qui tire parti de transactions de double dépense pour inférer une topologie, ne sont pas adaptés à notre travail.

En 2016, Neudecker et al. [NAH16] ont combiné une attaque passive et une attaque active pour inférer la topologie du réseau. L'hypothèse initiale de leur technique suppose qu'un nœud de supervision puisse être connecté à l'ensemble des nœuds du réseau P2P et d'observer les délais dans les propagations des messages. Pour atteindre de fortes précisions, cette technique doit être couplé avec une attaque active dans laquelle le nœud doit également forger des transactions et les propager pour étendre ses observations. L'hypothèse de l'attaque passive est très forte car un nœud n'a pas nécessairement de connexions entrantes disponibles, auquel cas le nœud devrait intervenir pour forcer une déconnexion sur le nœud cible.

En 2019, Delgado-Segura et al. [DBP+19] ont proposé une technique permettant de déduire la topologie en tirant parti des transactions orphelines et la structure de *mempool* (zone d'attente des transactions). L'idée est d'envoyer trois transactions spéciales (une transaction parente, une transaction flood et une transaction marqueur) à différents nœuds et de vérifier si la transaction marqueur peut être trouvée dans un autre nœud en envoyant un message `INV` (annonce de blocs/transactions). Les principaux problèmes de cette technique sont le coût et l'impact sur le réseau. Ils ont évalué à 400$ (en 2019) le coût de l'exécution de cette technique pour inférer la topologie du réseau à un moment donné. Ils ont également admis qu'ils n'ont pas exécuté cette technique sur le réseau principal de Bitcoin en raison de l'impact négatif que cette technique pourrait avoir. Pour ces deux raisons, cette technique n'est pas adaptée à nos travaux.

En 2018, Deshpande et al. [DBG18] et en 2020, Essaid et al. [EPJ19] [EPJ20] ont présenté une technique similaire pour déduire la topologie. Ils ont développé un émulateur et ont émulé une topologie probable du réseau, mais pas la topologie réelle. Nous ne savons pas dans quelle mesure cette topologie est probable, comment l'émulateur Bitcoin fonctionne ni dans quelle mesure il est proche d'un vrai client Bitcoin.

En 2015, Miller et al. [MLP+15] ont présenté leur méthodologie pour inférer les liens entre les pairs et déduire la topologie. Leur technique exploite le fait que pour les connexions sortantes, un nœud met à jour dans la table des contacts l'horodatage associé au nœud connecté à chaque fois qu'il reçoit un message de celui-ci. Ainsi, ils ont parcouru le réseau pour découvrir tous les nœuds du réseau et ont récupéré les horodatages de la liste de leurs contacts. Ils ont ensuite vérifié si les horodatages associés sont récents (moins de deux heures) et ont déduit les connexions entre les nœuds. Cette technique était efficace, mais les développeurs de Bitcoin [Thee] ont décidé de mettre en place une contre-mesure.

3.3.2 Évaluation de la technique utilisant les timestamps malgré les contre-mesures

Les développeurs de Bitcoin Core ont décidé de supprimer la mise à jour de l'horodatage lorsqu'un nœud reçoit des messages sur une connexion active [Thee]. Les mises à jour de l'horodatage pour d'autres critères sont conservées (par exemple lors de l'initiation d'une connexion entre deux nœuds).

Nous avons voulu évaluer cette contre-mesure et être sûrs que la topologie ne pouvait plus être déduite. Nous avons alors mené quelques expériences. Les premières étapes consistent à

TABLE 3.3 – Pourcentage de nouveaux nœuds découverts après n requêtes

	Nœud nº 1	Nœud nº 2	Nœud nº 3	Nœud nº 4	Nœud nº 5
60 requêtes	71.04%	74.31%	73.83%	75.66%	73.61%
120 requêtes	91,72%	93,61%	92,96%	94,04%	93,18%
180 requêtes	97,68%	98,38%	98,21%	98,56%	98,24%
240 requêtes	99,50%	99,63%	99,60%	99,67%	99,63%
300 requêtes	≈ 100%	≈ 100%	≈ 100%	≈ 100%	≈ 100%

parcourir la liste de contacts de chaque pair en envoyant de nombreux messages `GETADDR`. Comme chaque réponse contient des contacts aléatoires, il n'est pas possible de tous les découvrir mais nous pouvons tout de même obtenir une vue précise en multipliant les requêtes. Le graphique 3.11 montre le nombre de nouveaux contacts découverts à partir de nos cinq nœuds témoins en fonction du nombre de demandes effectuées. 180 demandes semblent suffisantes pour découvrir la plupart des contacts d'une liste, un plus grand nombre de messages n'entraîne qu'une très légère augmentation, comme le montre le tableau 3.3. Compte tenu du délai imposé entre les messages pour des raisons de sécurité du client, la plupart des contacts d'un nœud peuvent être découverts en deux heures environ. Ce délai pourrait être réduit en utilisant une approche distribuée, mais le faible taux d'attrition du réseau mis en évidence dans la section 3.2.5 ne justifie pas la nécessité de crawls plus rapides.

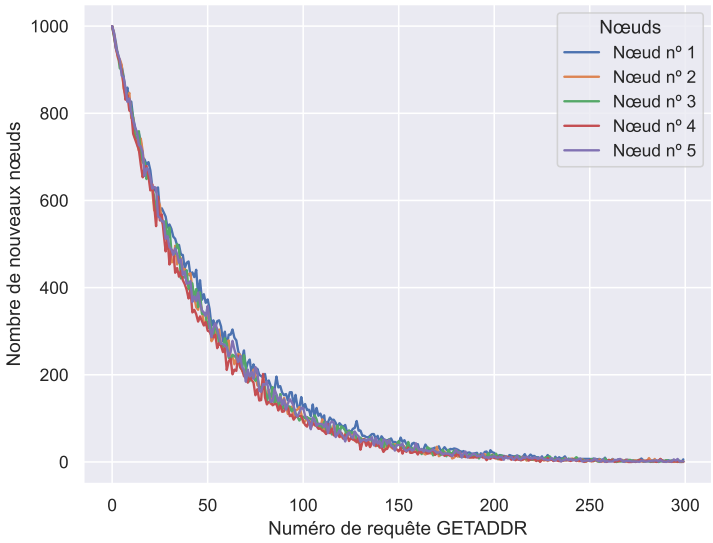

FIGURE 3.11 – Nombre de nouveaux nœuds découverts après n requêtes `GETADDR`

Nous avons ensuite voulu évaluer la possibilité de déduire les liens d'un pair en observant les horodatages rapportés pour chaque contact. Nous avons récupéré les listes de contacts de nos nœuds témoins et les avons analysées. Nous avons remarqué que les horodatages mis à jour ne sont pas assez récents pour utiliser la technique de Miller et al. Nous les avons analysés plus

en détail pour essayer de dévoiler un modèle dans la fréquence de mise à jour qui pourrait être utilisé pour déduire la topologie.

Dans une fenêtre temporelle de 24 heures, nous avons remarqué que les timestamps sont toujours mis à jour mais le nombre de mises à jour semble arbitraire. Pour rappel, par défaut un nœud Bitcoin peut avoir jusqu'à 8 connexions sortantes et 117 connexions entrantes. Nous pouvons voir dans le tableau 3.4 que le taux de faux positifs est très élevé (plus de 99 %) et même le taux de faux négatifs est inacceptable (50 %) lorsque nous considérons un trop grand nombre de mises à jour.

TABLE 3.4 – Nombre de connexions inférées en fonction du nombre de mises-à-jour du paramètre d'horodatage en 24h

	3 mises-à-jour	7 mises-à-jour	20 mises-à-jour
Vrais positifs	10	9	5
Faux positifs	9318	7039	2968
Vrais négatifs	44878	47157	51228
Faux négatifs	0	1	5

En conclusion, étant donné les contre-mesures mises en œuvre dans le client Bitcoin, il semble impossible aujourd'hui de déduire la topologie exacte du réseau. Les développeurs de Bitcoin ont même décidé de rendre plus difficile encore la déduction de la topologie en utilisant un cache pour les réponses aux messages `GETADDR` [Theb] (inclus en août 2020). Cela signifie qu'une même réponse peut être envoyée plusieurs fois pour des requêtes subséquentes et donc augmenter le temps pour découvrir l'ensemble des contacts d'un nœud.

Conclusion

Dans ce chapitre, nous avons présenté une analyse du réseau P2P des nœuds publics de Bitcoin et ses caractéristiques. Sur la base de mesures in-vivo sur un mois dont le jeu de données et le crawler utilisé pour le produire sont publics, nous avons montré que le réseau est stable, sans attaque Sybil évidente, et que les contre-mesures récentes contre l'inférence de la topologie sont efficaces. Mais il présente des propriétés plus inquiétantes comme le fait qu'une partie importante du réseau a tendance à mettre à jour la version du client lentement, ainsi que la popularité très déséquilibrée des pairs, même parmi ceux qui sont joignables.

En outre, nous avons étudié certains critères (churn, distribution des nœuds, clients utilisés) à surveiller grâce au crawler afin d'évaluer la bonne santé du réseau P2P dans le temps. Néanmoins, il est important de noter que l'analyse que nous avons présentée dans cet article concerne les nœuds publics (à savoir les nœuds complets accessibles) dans le réseau P2P Bitcoin et qu'elle ne peut pas être généralisée de manière triviale à tous les nœuds du réseau (portefeuilles, clients légers, mineurs). Ces nœuds publics sont néanmoins cruciaux, puisqu'ils agissent en tant qu'épine dorsale du réseau Bitcoin global : les portefeuilles et les clients légers ont besoin des nœuds publics complets pour se synchroniser avec le réseau et participer au protocole.

Le prochain chapitre vise à réaliser une analyse similaire du réseau P2P d'Ethereum. Cela permettra de comparer les caractéristiques des deux réseaux.

Chapitre 4

Caractéristiques du réseau P2P structuré d'Ethereum

Sommaire

Introduction

Nous avons vu dans le chapitre précédent les techniques de supervision pour un réseau non structuré comme Bitcoin. Ici, nous allons étudier et mettre en œuvre les techniques de supervision pour le réseau P2P structuré d'Ethereum. En effet, Ethereum utilise une table de hachage distribuée (*Distributed Hash Table* en anglais, ou DHT) pour construire un réseau d'*overlay*. Celui-ci est utilisé pour le protocole de découverte de nouveaux nœuds. Ethereum a implémenté une version légèrement modifiée de la DHT Kademlia [MM02]. Nous verrons les subtilités de ces modifications dans la suite de ce chapitre.

Pour la supervision de ce réseau P2P, nous avons conceptualisé et développé, à notre connaissance, le premier crawler open-source et récolté le premier jeu de données publiquement accessible pour ce réseau. Comme pour Bitcoin, cette étude a été réalisée pour évaluer la qualité du réseau P2P Ethereum vis-à-vis de certaines métriques impactant sa robustesse. Cette étude s'est déroulée sur deux périodes : tout le mois de septembre 2021 et entre le 15 février 2022 et le 7 mars 2022. Nous détaillerons dans la suite de ce chapitre, les caractéristiques du jeu de données récolté.

Enfin, nous introduisons également quelques nouvelles définitions propres à Ethereum. Le **NodeID** est la clé publique générée d'un nœud (format hexadécimal). Et un **enode** est l'association d'un NodeID, d'une adresse IP et d'un port (nodeid@ip :port).

4.1 Stratégie de supervision

4.1.1 Pile protocolaire d'Ethereum

Les développeurs d'Ethereum maintiennent un projet nommé « devp2p » [The21] qui contient la spécification des différents protocoles utilisés dans le réseau P2P d'Ethereum. Le client officiel Go-Ethereum (également appelé Geth) ainsi que d'autres clients, notamment OpenEthereum (anciennement Parity-Ethereum), mettent en œuvre cette spécification [Thek, Ope]. Dans le schéma 4.1, nous donnons un aperçu d'une interaction classique entre deux pairs qui s'appuie sur les trois protocoles suivants : « Node Discovery », « RLPx » et le protocole « Ethereum Wire ». Dans ce chapitre, nous allons nous concentrer sur le protocole « Node Discovery » qui est utilisé par notre crawler pour contacter les pairs et en faire l'inventaire. Les protocoles « RLPx » et « Ethereum Wire » ne seront pas détaillés ici car notre crawler ne les implante pas. Pour résumer, le protocole « RLPx » permet l'établissement d'une clé de chiffrement entre deux nœuds pour permettre le chiffrement des communications, et le protocole « Ethereum Wire » permet ensuite à deux nœuds de s'échanger des messages applicatifs, tel que l'envoi d'un nouveau bloc par exemple.

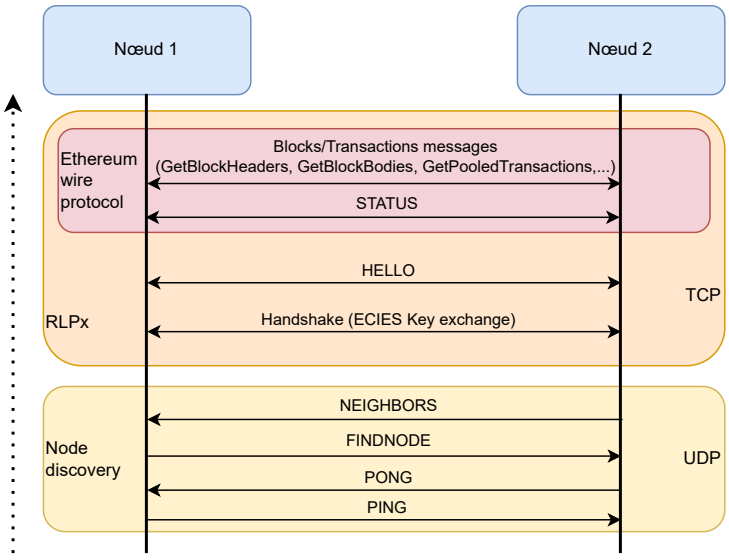

FIGURE 4.1 – Vue d'ensemble des couches protocolaires d'Ethereum utilisées entre deux pairs du réseau

Le protocole de découverte de nœuds (« Node Discovery ») est basé sur une implémentation modifiée de la DHT Kademlia [MM02]. Les principaux avantages de Kademlia sont sa simplicité d'implémentation et son algorithme de routage très efficace et évolutif dont la complexité est $\mathcal{O}(\log_2 N)$, N étant le nombre de nœuds du réseau. Nous rappellerons d'abord les grands principes de Kademlia (voir la section 1.2 de l'état de l'art pour plus de détails), puis nous soulignerons les différences de l'implémentation d'Ethereum.

Le but de Kademlia, dans un système de partage de fichier distribué, est de permettre la récupération efficace des données qui sont distribuées entre les pairs grâce à leurs clés (hashes). Un pair est représenté par un identifiant de nœud (Node ID) généré aléatoirement, et les données stockées sont représentées par leur hash (par exemple en utilisant l'algorithme SHA-1), tous deux étant des valeurs de 160 bits. Kademlia utilise une métrique XOR pour calculer la distance entre deux nœuds ou entre un nœud et une donnée dans l'espace d'adressage DHT. Pour être évolutive, la table de routage d'un pair ne stocke qu'un nombre limité de nœuds (contacts) organisés en K-buckets (groupe de K-contacts). Chaque bucket couvre une partie de l'espace d'adressage suivant une distribution logarithmique : les K contacts suivants couvrent la moitié de l'espace par rapport au bucket précédent. Pour être plus précis, les groupes de K-contacts sont organisés de manière à ce que chaque bucket i détienne les informations des pairs dont la distance est comprise entre 2^{160-i} et $2^{160-(i+1)}$ par rapport au Node ID du pair courant. Cette structure garantit que tout identifiant (pair ou donnée) peut être trouvé en un nombre limité de sauts. Les données peuvent être stockées sur des nœuds dont la distance est inférieure à un seuil (c'est-à-dire dans une zone de tolérance). Pour faire face au churn (c'est-à-dire l'arrivée et départ continus de pairs), les données doivent également être répliquées sur plusieurs nœuds. Enfin, un nœud peut demander des données aux nœuds responsables de leur stockage. Un nœud maintient une table de routage qui est divisée en 160 buckets (contenant au maximum `k` nœuds) basés sur la métrique XOR entre son propre ID de nœud et l'ID de nœud d'un autre pair. De cette façon, il peut stocker un morceau de données dans un nœud dont l'ID est proche du hachage des données.

4.1.2 Implantation de Kademlia dans Ethereum

Comme Kademlia, le protocole de découverte des nœuds d'Ethereum est basé sur UDP et utilise la métrique XOR pour calculer la distance entre les pairs. Mais il existe des différences significatives entre la conception originale de Kademlia et l'implémentation Ethereum de la DHT. Tout d'abord, et c'est important de le préciser, Ethereum n'utilise pas la DHT pour le stockage et la récupération de données, mais tous les pairs stockent l'intégralité de la blockchain. La DHT d'Ethereum est uniquement utilisée pour la découverte de nœuds. De même, un pair est identifié par une clé publique ECDSA (Elliptic Curve Digital Signature Algorithm) de 512 bits (contre une ID de 160 bits dans Kademlia) et la fonction XOR est appliquée au hachage de cette clé. La fonction de hachage utilisée est Keccak256 et non la norme SHA3 FIPS 202 du National Institute of Standards and Technology (NIST) qui est basée sur l'algorithme Keccak. Essentiellement, la norme diffère sur les paramètres de padding, ce qui donne deux hachés différents. Également, il ne s'agit pas simplement de l'application de l'opération XOR entre ces hachés, Ethereum utilise plutôt le résultat tronqué de l'opération $\log_2$(a XOR b) comme fonction de distance.

En ce qui concerne les primitives du réseau, il existe 4 types de paquets utilisés pour mettre en œuvre le protocole de découverte de nœuds : `PING`, `PONG`, `FINDNODE` et `NEIGHBORS`. Lorsqu'un message `PING` est reçu, le destinataire doit répondre par un message `PONG`. L'échange `PING-PONG` est également utilisé pour obtenir une *endpoint proof* : pour prévenir les attaques de spoofing et d'amplification de trafic, un nœud ne doit répondre qu'à un expéditeur dont la réactivité (à un message `PING`) a été vérifiée. Le mécanisme de découverte de nœuds est construit autour du paquet FINDNODE : il demande des informations sur les nœuds proches d'un paramètre donné « ID de la cible ». Le destinataire de ce message doit répondre par un paquet NEIGHBORS contenant les seize nœuds les plus proches de la cible connus à partir de ses propres buckets. Il convient de noter que l'identifiant utilisé dans les paquets du protocole de découverte de nœuds est la clé publique ECDSA de 512 bits au lieu de sa version haché qui fait, elle, 256 bits, bien que la première action du destinataire soit de refaire le hachage. Cela n'est pas optimal du point de

vue du réseau. Comme pour Bitcoin, les transactions et les blocs sont échangés selon un protocole de gossip grâce à un autre niveau de réseau d'*overlay* non structuré constitué de connexions TCP directes établies entre pairs (voir la couche RPLx dans la figure 4.1), sans aucune considération pour leur place dans la DHT. La primitive FINDNODE est similaire à celle de Kademlia, mais à la différence d'Ethereum son objectif principal est plutôt de localiser un pair qui détient les données recherchées.

4.1.3 Methodologie de supervision

Nous avons développé un crawler, nommé Crawleth, pour explorer le réseau P2P d'Ethereum. Crawleth fonctionne au niveau du protocole Node Discovery. Il est écrit en Python et se base en partie sur le client Trinity Ethereum [18]. Son code est open source et disponible publiquement sur le Gitlab de notre institut de recherche [Eisa] [19]. L'objectif de notre outil est de trouver tous les nœuds qui participent au protocole Node Discovery d'Ethereum, quel que soit le protocole utilisé ultérieurement : il peut trouver les nœuds du réseau principal et du réseau de test d'Ethereum, ainsi que les nœuds qui participent à d'autres systèmes de blockchain construits sur la couche Node Discovery d'Ethereum. Mais des études antérieures [KMM+18] ont montré que la majorité des pairs qui suit le protocole Node Discovery appartient en fait au mainnet d'Ethereum.

Il est important de noter que le protocole dispose d'une contre-mesure contre les attaques par amplification de trafic : il vérifie que l'expéditeur d'un paquet participe au protocole de découverte. L'expéditeur est considéré comme vérifié s'il a répondu à un `PING` au cours des 12 dernières heures (comme expliqué à la section 4.1.2). Ainsi, notre crawler écoute constamment les messages `PING` et répond avec un message `PONG` correspondant.

La méthodologie d'exploration de notre logiciel est décrite dans la figure 4.2. Tout comme Bitcoin, des nœuds spéciaux (aussi appelés nœuds d'amorçage ou en anglais *bootstrap nodes*, abrégé en *bootnodes* chez Ethereum) sont déployés pour permettre à un nouveau nœud de rejoindre le réseau P2P. Pour commencer, Crawleth teste la connectivité des nœuds d'amorçage (dont les adresses IP et les ID de nœuds ont été récupérés dans le code source du client Geth où ils sont pré-configurés) en envoyant un paquet `PING` et en attendant la réponse `PONG`. Ensuite, pour chaque nœud qui a répondu, il envoie un paquet `FINDNODE` dont le paramètre est l'ID du nœud d'amorçage lui-même. Les requêtes `FINDNODE` utilisées par notre crawler sont toujours centrées sur le Node ID contacté, où sa table de routage a la plus grande précision car les buckets sont les plus profonds (plus de contacts sont connus). Le destinataire répondra avec un paquet `NEIGHBORS` contenant seize nœuds proches de lui, en termes de distance XOR entre les identifiants de nœuds.

Crawleth poursuivra cette stratégie avec les nœuds nouvellement découverts jusqu'à ce qu'il n'y ait plus de nouveaux nœuds à découvrir, ce qui signifie qu'il a parcouru l'ensemble de l'espace d'adressage de la DHT. Quand un crawl est terminé, il exporte les informations des nœuds découverts (adresse IP, port UDP, Node IDs, géolocalisation, numéro d'AS) et commence un nouveau crawl.

4.1.4 Configuration du serveur

La machine que nous avons utilisée est un ordinateur sous Ubuntu 18.04 LTS équipé d'un processeur Intel Xeon E5-2420 v2 6 cœurs, de 16 Go de RAM et d'une liaison réseau 1 Gb/s. Pour gérer le grand nombre de connexions nécessaires, nous avons dû augmenter la limite de fichiers ouverts de notre système GNU/Linux à 1 000 000. La machine a fait fonctionner le nœud

18. `https://github.com/ethereum/trinity`
19. `https://gitlab.inria.fr/jeisenba/Crawleth`

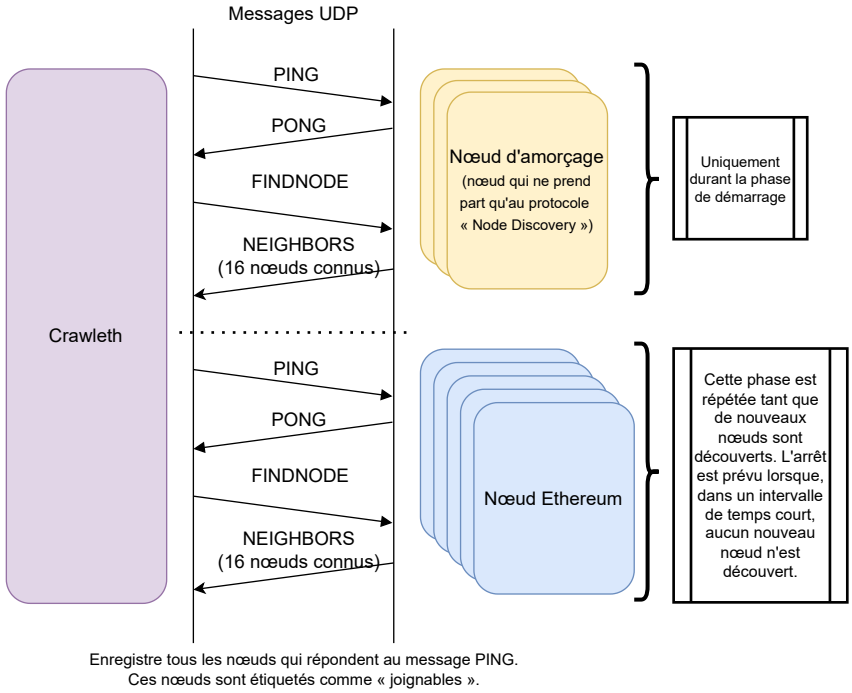

FIGURE 4.2 – Stratégie d'exploration de Crawleth

de crawling 24 heures sur 24, 7 jours sur 7 pendant les deux périodes de crawl, qui, pour rappel, s'étendent durant tout le mois de septembre 2021 pour la première et du 15 février 2022 au 7 mars 2022 pour la deuxième période.

Il convient de mentionner que notre infrastructure ne disposait pas d'une liaison IPv6 et d'un proxy TOR, de sorte que tous les nœuds découverts sont des nœuds IPv4. Néanmoins, comme nous le verrons dans la section 4.1.6 d'autres outils comme Ethernodes ou Etherscan NodeTracker n'intègrent pas non plus de tels nœuds. Comme leurs méthodologies ne sont pas connues, nous ne pouvons pas affirmer qu'il n'y a pas de nœuds IPv6 dans le réseau, ils peuvent simplement manquer d'un lien IPv6 également. En ce qui concerne Tor, il semble impossible à première vue de trouver un nœud utilisant le réseau Tor car celui-ci utilise exclusivement le protocole de transport TCP, alors qu'Ethereum utilise UDP pour la découverte de nœuds. Bien qu'il soit possible d'amorcer manuellement un nœud avec des pairs connus du réseau et de désactiver le mécanisme de découverte (*Node Discovery Protocol*), il est très peu probable que cela se produise car cette solution de contournement nécessite une gestion manuelle fastidieuse des contacts pour maintenir le nœud connecté.

4.1.5 Jeux de données collectés

Le crawler est actif depuis aout 2021 et nous permet d'avoir les informations du réseau P2P en temps réel. Pour cette thèse, nous avons analysé deux périodes, à savoir septembre 2021 et entre le 15 février 2022 et le 7 mars 2022 [20]. Dans la section 4.2, nous détaillons les raisons pour lesquelles nous avons choisi ces deux périodes pour notre étude. Un extrait de ces jeux de données est présenté en annexe B.

Ils se présentent sous la forme de fichier `JSON` dont le format est le suivant :

20. Disponible ici `https://concordia-eth-p2p.lhs.loria.fr/index.html`

- « stats » est l'objet qui contient des informations générales du crawl en question ;
  - « aggregated_crawls » est le nombre de crawls agrégés au sein du fichier ;
  - « original_filenames » contient les noms des fichiers des crawls agrégés (date au format UTC+0) ;
  - « up_count » est le nombre de nœuds en ligne découvert par notre outil ;
- « up » est la liste des nœuds récupérée par Crawleth, elle contient plusieurs objets ;
  - « IP address » est l'adresse IP anonymisée d'un nœud ;
  - « UDP port » est le numéro de port UDP utilisé ;
  - « Seen node IDs » est la liste contenant l'ensemble des NODE IDs trouvé pour ce nœud (IP-port) ;
  - « as » le nom de l'AS associé à l'adresse IP ;
  - « asn » le numéro de l'AS ;
  - les cinq derniers (« city », « country », « country_code », « latitude » et « longitude ») correspondent aux données de géolocalisation de l'adresse IP d'après la base de données GeoLite 2 de MaxMind [Max].

4.1.6 Validation

Afin de valider la justesse et la qualité de notre outil, nous allons détailler dans les deux prochaines sous-sections respectivement les mesures de convergence que nous avons effectuées puis comparer la couverture de Crawleth avec des sources externes indépendantes.

Validation interne L'exploration de la DHT, qui consiste à demander successivement tous les nœuds de l'espace d'adressage, devrait converger de manière remarquable : présenter un taux de découverte plutôt constant de nouveaux nœuds pendant l'exploration alors que le crawler avance dans l'espace d'adressage et, à la fin, une chute soudaine du taux de découverte lorsque tout l'espace d'adressage a été couvert et qu'il n'y a plus de nouveaux nœuds à découvrir.

Comme nous pouvons le voir sur les figures 4.3 et 4.4, il y a en fait trois phases pendant un crawl. La première phase est courte (moins d'une minute) et est considérée comme une phase d'amorçage où le crawler découvre un nombre élevé de nouveaux nœuds par seconde. Ensuite, la phase principale dure pendant presque tout le crawl et présente une croissance régulière où il découvre un nombre moyen constant de nouveaux nœuds par seconde qui correspond à une croissance linéaire tandis que le crawler progresse dans l'espace d'adressage de la DHT. Enfin, et comme prévu, il se termine par une dernière phase courte où il n'y a aucun nouveau nœud à découvrir.

En outre, nous avons déployé cinq nœuds témoins indépendants exécutant le client officiel Go Ethereum non modifié [Thek] (Geth v1.10.3, publié en mai 2021). L'un d'entre eux a été configuré pour rejoindre réseau de test Ropsten d'Ethereum (dont l'identifiant de réseau est 3) et les quatre autres ont été configurés pour rejoindre le réseau principal (*mainnet*) d'Ethereum (dont l'identifiant de réseau est 1). Tous les autres paramètres réseau ont été laissés par défaut, en particulier le nombre maximum de connexions simultanées qui est de 50. Tous les nœuds ont été déployés au sein du même serveur mais cela n'affecte pas leurs connexions puisque la découverte des nœuds utilise un réseau d'overlay (DHT) qui est totalement agnostique aux considérations géographiques ou du niveau de la couche IP. Ils ont tous des NodeIDs différents, donc une place différente dans l'espace d'adressage DHT. Lorsqu'ils se connectent au réseau P2P, ils reçoivent tous des réponses différentes de la part des nœuds d'amorçage. Nous avons également vérifié qu'ils ont des connexions actives différentes. En conclusion de cette expérience de validation, Crawleth a été capable de trouver tous nos nœuds témoins lors des explorations du réseau P2P.

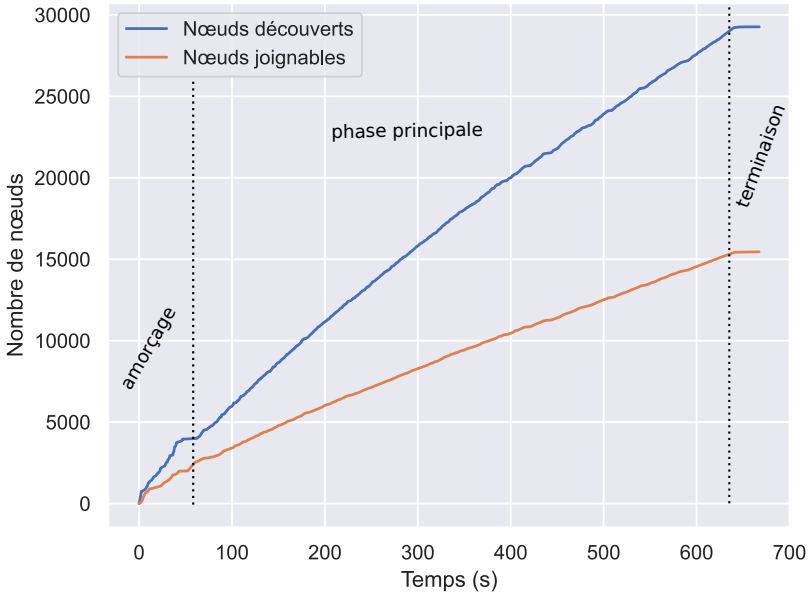

FIGURE 4.3 – Nombre cumulé de nœuds découverts et accessibles au cours d'un crawl

Validation externe Pour évaluer la qualité de notre programme d'exploration, nous avons voulu comparer sa couverture avec d'autres outils disponibles. Comme les autres outils ne sont pas documentés, ce résultat ne constitue pas une preuve mais plutôt un indice. Cette comparaison est résumée dans le Tableau 4.1. Il existe deux sites Web, Ethernodes [Etha] [21] et Etherscan [Ethc] [22], qui proposent des statistiques sur le réseau Ethereum. À notre connaissance, ils utilisent tous deux leur propre outil qui n'est pas open source et leur méthodologie n'est pas documentée. En septembre 2021, Ethernodes et Etherscan ont pu trouver un nombre similaire de nœuds (que nous supposons publiquement joignables, à défaut de documentation sur la méthodologie employée) : 3 700 pour le premier et 3 000 pour le second. En février et mars 2022, Ethernodes a pu trouver environ 6 000 nœuds et Etherscan 2 500 nœuds. En comparaison, Crawleth pouvait trouver en moyenne plus de 16 000 nœuds publiquement joignable et pouvait découvrir environ 30 000 nœuds au total (accessibles et non accessibles) en septembre 2021. Lors de l'augmentation soudaine du nombre de nœuds découverts en février 2022, Crawleth a pu trouver en moyenne plus de 33 000 nœuds accessibles au public, avec un pic de 42 000 nœuds, et a pu découvrir plus de 50 000 nœuds (accessibles et inaccessibles) au cours d'un seul crawl. Il est important de noter que les deux sites web ont présenté une baisse significative de leurs chiffres depuis mars/avril 2021. Auparavant, Ethernodes et Etherscan comptaient environ 10 000 nœuds. Nous pouvons émettre l'hypothèse qu'ils ont changé la façon dont ils recensent les nœuds, par exemple en ignorant les nœuds qui ne font pas partie du mainnet Ethereum, mais nous ne pouvons pas en être sûrs en raison du manque de documentation officielle.

Nous pouvons également comparer la couverture de Crawleth avec les travaux de Kim et al. [KMM+18] sur NodeFinder , Gao et al. [GSW+19] et Maeng et al. [MEJ20]. En avril 2018, NodeFinder observait environ 15 500 nœuds actifs tandis qu'en 2019, Gao et al. ont signalé 11 000 nœuds actifs et Maeng et al. [MEJ20] ont observé avec leur outil en 2020 8223 nœuds actifs. Même si plusieurs années séparent les crawls, les chiffres observés sont du même ordre de

21. `https://ethernodes.org/`
22. `http://etherscan.io/nodetracker`

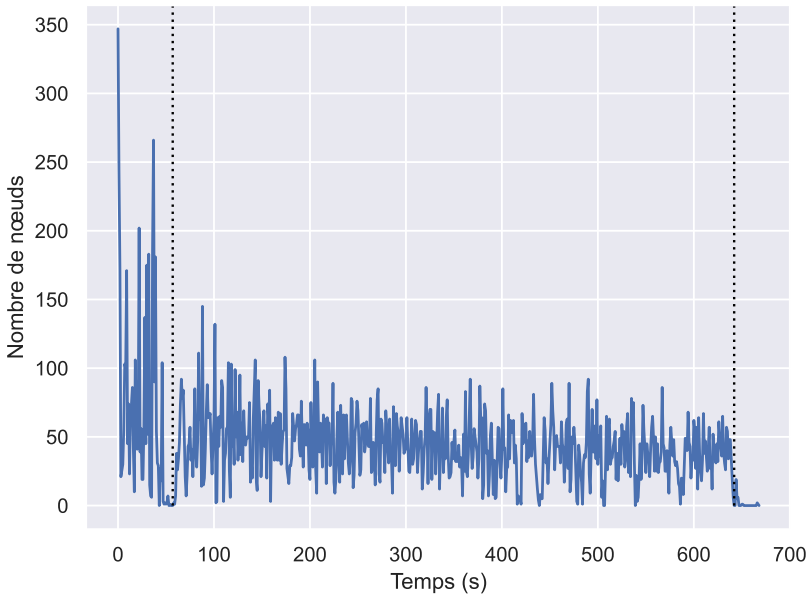

FIGURE 4.4 – Nombre de nœuds découverts par seconde au cours d'un crawl

grandeur si l'on considère la période antérieure à février 2022. De plus, ces travaux ont recensés les nœuds complets (*fullnode*) alors que Crawleth recense la totalité des nœuds du réseau.

Sur la base de la validation interne et externe, la méthodologie de Crawleth peut raisonnablement être considérée comme juste et précise pour trouver tous les nœuds qui composent le réseau P2P Ethereum.

4.2 Métrologie réseau

Il existe plusieurs types de mesures qui caractérisent un réseau P2P. Nous avons voulu être le plus exhaustif possible, en présentant le nombre de nœuds qui composent le réseau, leur distribution géographique et logique (chez des particuliers ou fournisseur de cloud) ainsi que le ratio de connexions/déconnexions des pairs. La topologie exacte du réseau est une caractéristique très difficile à inférer, car c'est une information que les développeurs des clients Ethereum veulent garder la plus secrète possible et donc ils développent des contremesures efficaces pour empêcher cette inférence, à l'instar de Bitcoin (voir section 3.3).

Les analyses concernent les données recueillies par notre crawler durant deux périodes significatives : le mois de septembre 2021 et entre le 15 février et le 7 mars 2022. En dehors de ces périodes, le crawler n'est pas stoppé et nous permet d'avoir, en temps réel, des caractéristiques que nous étudierons dans le chapitre 6. Le mois de septembre 2021 est représentatif de l'état du réseau depuis le déploiement de notre crawler. Et en février 2022, nous avons observé une augmentation soudaine et massive du nombre de nœuds qui mérite une attention particulière. Nos analyses tiennent compte des caractéristiques du réseau qui peuvent avoir un impact sur la fiabilité. Les deux ensembles de données sont rendus publics [Eis21] avec la seule limitation que les adresses IP ont dû être pseudo-anonymisées suivant les recommandations de l'ICANN [Int18] qui offre un compromis acceptable entre utilité et confidentialité. Plus précisément, tout comme pour le jeu de données de nœuds Bitcoin, un hachage salé est fourni et le dernier octet de l'adresse

TABLE 4.1 – Comparaison des tailles de réseau P2P trouvé par Crawleth et d'autres outils externes et indépendants

	Date	Taille moyenne du réseau
Crawleth	Février–Mars 2022	33 000
Ethernodes [Etha]	Février–Mars 2022	6 000
Etherscan [Ethc]	Février–Mars 2022	2 500
Crawleth	Septembre 2021	16 600
Ethernodes	Septembre 2021	3 700
Etherscan	Septembre 2021	3 000
Ethernodes	Février 2021	12 000
Etherscan	Février 2021	8 000
Maeng et al. [MEJ20]	2 et 3 mars 2020	8 223
Gao et al. [GSW+19]	Janvier 2019	11 000
NodeFinder [KMM+18]	Avril 2018	15 454

IP est mis à 0. Les scripts utilisés pour effectuer l'analyse sont également disponibles dans notre dépôt git [Eisa], rendant l'ensemble de l'étude entièrement reproductible. Nous proposons en annexe B un extrait de ces fichiers de données.

4.2.1 Nombre de nœuds et distribution géographique

La première métrique que nous pouvons déduire de nos ensembles de données est la taille du réseau et la géolocalisation des nœuds. Comme nous pouvons le voir sur la figure 4.5, la taille du réseau a été assez stable pendant 7 mois (entre juillet 2021 et janvier 2022) depuis le déploiement du crawler avec une croissance lente mais régulière faisant passer le nombre de nœuds de 16 000 à 22 000 sur la période. Ensuite, le nombre de nœuds a rapidement augmenté le 23 février 2022 pour atteindre un maximum éphémère d'environ 42 000 nœuds, puis a diminué pour atteindre un nouveau plateau à 32 000 nœuds en début mars. Certains résultats numériques notables peuvent également être trouvés dans le tableau 4.1.

Au total, en incluant les nœuds inaccessibles mais présents dans les réponses *NEIGHBORS*, Crawleth a découvert environ 30 000 nœuds par crawl en septembre 2021, contre 16 000 nœuds accessibles. Il est également important de noter que le crawler a pu trouver au total 439 561 identifiants de nœuds associés à 103 332 associations uniques d'adresse IP à un port au cours du mois de mesure. La taille de ce réseau est du même ordre de grandeur que celui de Bitcoin, mais en comparaison avec d'autres réseaux P2P de la littérature, en particulier les réseaux P2P de partage de fichiers comme Gnutella, KAD ou Bittorent qui peuvent facilement rassembler des centaines de milliers de nœuds, il reste relativement modeste.

De plus, nous avons constaté que le réseau était assez bien équilibré à travers le monde en septembre 2021, malgré un peu plus de nœuds situés en Europe qu'en Amérique du Nord ou

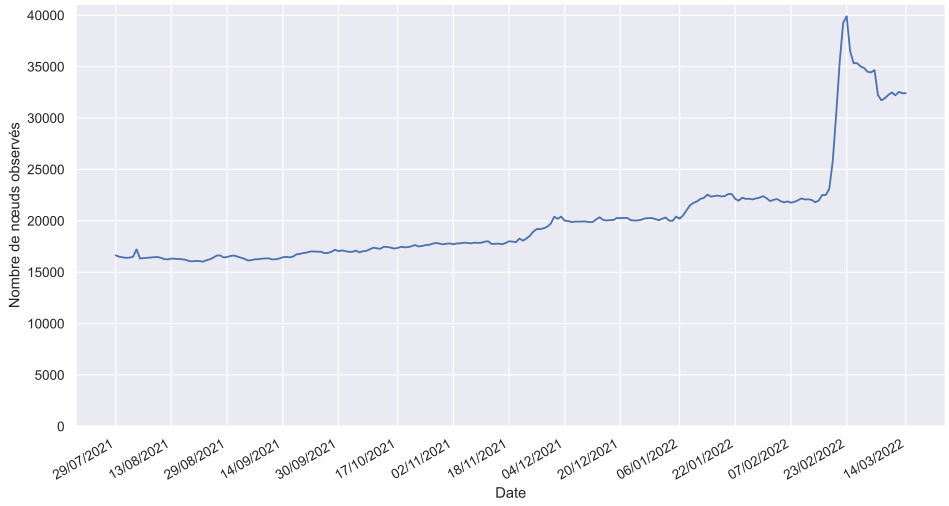

FIGURE 4.5 – Nombre de nœuds Ethereum joignables

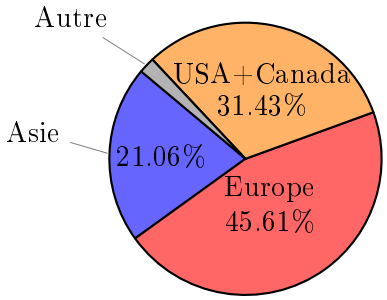

FIGURE 4.6 – Géolocalisation des nœuds publics en septembre 2021

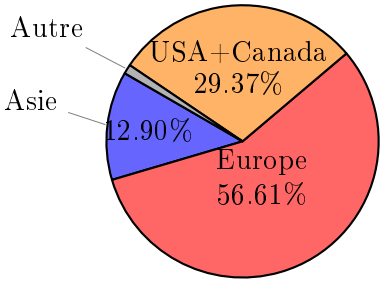

FIGURE 4.7 – Géolocalisation des nœuds publics durant la période févr.–mar. 2022

en Asie, comme nous pouvons le voir sur le graphique 4.6. Cependant, comme le montre le graphique 4.7, les pairs qui ont rejoint le réseau en février 2022 ont modifié l'équilibre : le réseau est désormais davantage concentré en Europe au détriment de l'Asie. À noter que la localisation des nœuds n'est pas nécessairement corrélée à la localisation de la puissance de calcul déployée par les mineurs. Nous avons rappelé dans le chapitre 3 que des disparités existent à cet égard dans Bitcoin et des comportements très similaires peuvent être légitimement attendus pour Ethereum. Malheureusement, le manque d'étude estimant la géolocalisation des mining pools d'Ethereum nous empêche de tirer une conclusion claire sur cet aspect.

4.2.2 Distribution entre ASes

En septembre 2021, 60 % des adresses IP d'un nœud étaient hébergées dans seulement 10 systèmes autonomes ((*Autonomous systems* en anglais, abrégé ASes)) sur 2 257 (0,44 %). En février et mars 2022, c'est encore plus centralisé puisque 71% des adresses IP étaient hébergées dans 10 AS sur 2 292 (0,44%). Neuf de ces 10 AS sont les mêmes au cours des deux périodes et un seul d'entre eux n'est pas un fournisseur de cloud computing. Parmi ces fournisseurs cloud, nous pouvons citer, par ordre d'importance en termes de centralisation de nœuds :

— Amazon (entreprise américaine) ;
— Hetzner (allemand) ;
— DigitalOcean (américain) ;
— Contabo (allemand) ;
— Google (américain) ;
— OVH (France) ;
— Alibaba US (gouvernance chinoise mais hébergé aux États-Unis) ;
— Microsoft (américain) ;

Nous pouvons voir que la majorité des fournisseurs cloud de ce classement est américaine, cela dénote une centralisation encore plus importante car ces fournisseurs sont soumis aux lois des États-Unis et doivent donc se conformer à ceux-ci en cas de censure, surveillance, coupure du trafic Ethereum. Il y a alors un paradoxe entre la philosophie de décentralisation prôné par la fondation Ethereum (et intrinsèque par définition à un réseau P2P en bonne santé) et la centralisation effective observée.

En conclusion, même si la répartition géographique des nœuds est bien équilibrée, nous avons observé que la plupart des nœuds sont hébergés dans un très petit nombre d'AS. Cela dénote une centralisation du réseau qui n'est pas une bonne propriété pour sa résilience. De plus, le nombre important de pairs qui ont rejoint le réseau en février 2022 a considérablement accentué la concentration géographique et celle du réseau.

4.2.3 Taux d'attrition (*churn rate*)

Une métrique intéressante du réseau à analyser est le ratio de connexion et de déconnexion des nœuds accessibles. Elle dénote la stabilité du réseau. Un crawl dure environ 15 minutes, donc pour observer le taux de renouvellement horaire, nous avons agrégé 4 ou 5 crawls (en fonction de leur durée effective) pour obtenir un instantané du réseau sur chaque heure. Pour des raisons de lisibilité, la figure 4.8 ne représente le taux de désabonnement horaire que du 27 février 2022 au 2 mars 2022, mais la tendance est la même sur toute la période de mesure. Les taux de déconnexion et de (re)connexion par heure et par jour sont assez constants et stables, respectivement autour de 4 % et 18 %. Ces taux sont considérés comme faibles par rapport aux normes des réseaux P2P [Cho11], ce qui signifie que le réseau peut être considéré comme stable. Même si une petite partie des nœuds a tendance à se déconnecter souvent, la taille du réseau ne change pas à mesure que les nœuds se reconnectent ou sont remplacés par d'autres. Pour l'ensemble du mois de septembre 2021, le taux de renouvellement quotidien est d'environ 20 % en moyenne.

En outre, il est important de noter que notre crawler ne fait pas la différence entre les différentes blockchains qui reposent sur le protocole de découverte des nœuds d'Ethereum. En particulier, le "réseau de test" de l'instance Ethereum est probablement moins stable que le réseau principal. D'autres systèmes de blockchain construits au-dessus du protocole de découverte des nœuds d'Ethereum et moins populaires qu'Ethereum peuvent également être moins stables. Mais comme l'ont montré d'autres études, notamment [KMM+18], le réseau est très majoritairement composé de nœuds du Mainnet.

4.3 Comparaison des caractéristiques des réseaux P2P Bitcoin et Ethereum

Bitcoin et Ethereum sont souvent comparées, car ce sont les deux cryptomonnaies les plus populaires. Les réseaux P2P de ces deux blockchains sont fondamentalement différents, car Bit-

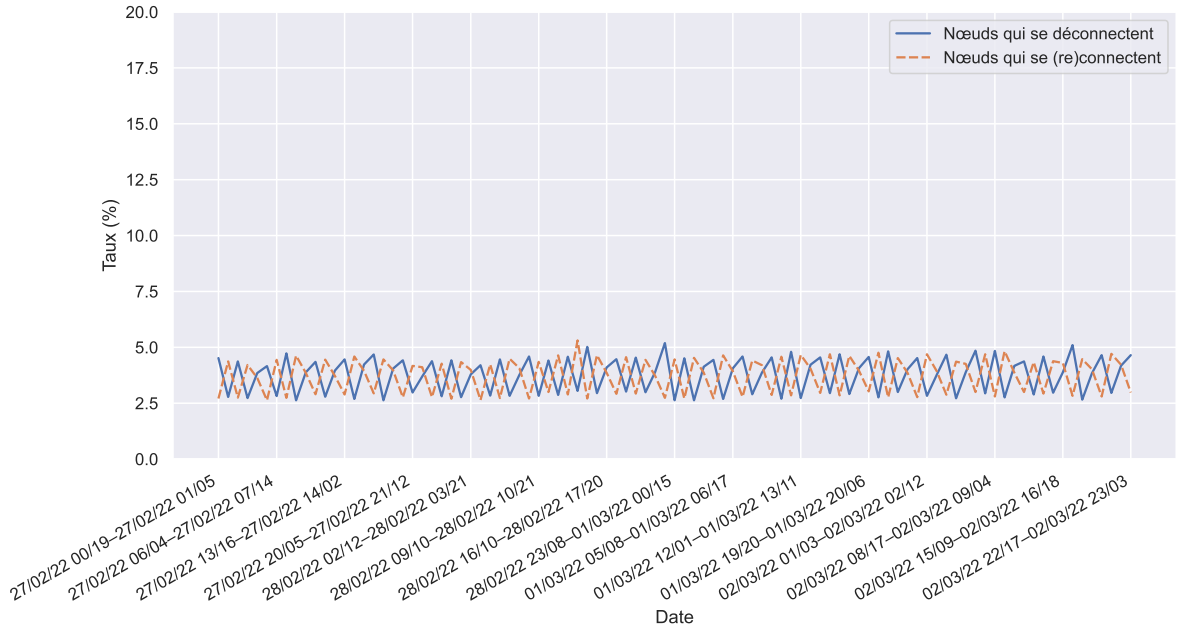

FIGURE 4.8 – Taux de connexion/déconnexion horaire des nœuds joignables d'Ethereum

coin repose sur un réseau non structuré et Ethereum repose sur un réseau structuré utilisant une DHT. Dans la pratique, les deux réseaux P2P utilisent une propagation des blocs et transactions par inondation (*gossip*) ce qui dénote plutôt une forte similarité de fonctionnement des deux réseaux. Dans le chapitre 3 et dans ce présent chapitre (4), nous avons analysé en détail diverses métriques pouvant impacter la fiabilité de ces deux réseaux P2P. Dans cette section, nous allons résumer brièvement et comparer les informations importantes que nous avons découvertes sur Bitcoin et Ethereum. Ces informations sont également résumées dans le tableau 4.2.

Taille des réseaux

Les deux réseaux ont des tailles dont l'ordre de grandeur est similaire, à savoir la dizaine de milliers de pairs. Bitcoin oscille autour de la barre symbolique des 10 000 nœuds alors que les pairs sur Ethereum sont plus de deux fois plus nombreux et plutôt autour de 25 000 nœuds.

Ces deux tailles sont plutôt correctes pour rendre plus difficiles et coûteuses des attaques comme l'attaque Eclipse même si historiquement les réseaux P2P de partage de fichiers pouvaient compter jusqu'à plusieurs millions de pairs. Comparativement à d'autres blockchains qui reposent sur des nœuds validateurs (par exemple Stellar) et dont la taille dépasse à peine la centaine de nœuds, Bitcoin et Ethereum sont plus résilients à cet égard.

Distribution géographique

Les deux réseaux se ressemblent également concernant la distribution géographique des pairs. Les pairs sont essentiellement présents en Europe, à plus de 40% tant pour Bitcoin que pour Ethereum. L'Amérique du Nord arrive en deuxième et l'Asie est dernière. La distribution est plutôt saine et ne dénote pas de concentration géographique des pairs dans un pays en particulier.

Churn

Concernant le churn, les deux réseaux se démarquent un peu, Bitcoin a un churn journalier de l'ordre de 5-6% tandis que les pairs d'Ethereum ont plutôt un churn journalier de l'ordre de 18%. Même si la différence semble significative, le churn d'Ethereum n'est pas considéré comme particulièrement haut. On peut plutôt considérer que le churn de Bitcoin est particulièrement bas, ce qui dénote une grande stabilité dans les pairs du réseau.

Distribution entre ASes

Les pairs des deux réseaux sont essentiellement hébergés chez des fournisseurs de cloud, mais la différence notable se trouve chez Ethereum dont les pairs sont très centralisés dans une dizaine d'hébergeurs. En effet, 70% des pairs sont centralisés chez 10 AS dont 9 sont des fournisseurs de cloud. Tandis que pour Bitcoin, ce nombre descend à 35% des pairs. Ethereum souffre d'une grande concentration des pairs, ce qui affecte sa capacité à résister à des attaques qui pourraient cibler spécifiquement quelques ASes ou même résister à des mesures de censure à l'échelle d'un pays dont les ASes sont obligés de se soumettre aux lois locales.

TABLE 4.2 – Tableau de synthèse des propriétés des réseaux P2P de Bitcoin et d'Ethereum.

	Bitcoin	**Ethereum**
Nombre de nœuds	∼10 000	∼25 000
Distribution géographique	Équilibrée	
Churn journalier	∼5-6%	∼18%
Distribution entre ASes	35% des nœuds concentrés dans 10 ASes	70% des nœuds concentrés dans 10 ASes

Conclusion

Dans ce chapitre, nous avons d'abord présenté notre crawler open source pour le réseau P2P Ethereum. Nous avons démontré sa qualité et analysé deux ensembles de données d'un mois que nous avons également publiés. Les résultats montrent que le réseau P2P Ethereum présente de bonnes propriétés telles qu'une distribution géographique correcte et un taux d'attrition modéré, mais aussi des propriétés plus préoccupantes comme la concentration des nœuds dans quelques ASes. La taille du réseau est restée stable pendant plusieurs mois, mais a soudainement augmenté pour atteindre, à son apogée, le double de la taille du réseau initialement mesurée, et ce, sans explication évidente ni modèle qui pourrait caractériser les nouveaux arrivants. Nous n'avons pas été en mesure d'expliquer cette augmentation soudaine en corrélant ces arrivées massives avec un événement extérieur à la communauté blockchain.

Notre analyse du réseau P2P d'Ethereum nous a amené à constater que sa DHT, bien que pleinement fonctionnelle, est largement inexploitée, car n'indexant aucune donnée. Le prochain chapitre montre comment le fonctionnement d'Ethereum pourrait être optimisé pour tirer parti de cette structure.

Troisième partie

Stockage efficace des données et prévention d'attaques Sybil sur Ethereum

Chapitre 5

Stockage efficace des données de la blockchain Ethereum dans la DHT

Sommaire

Introduction

Dans le chapitre précédent, nous avons étudié les caractéristiques du réseau P2P d'Ethereum et nous avons notamment mis en exergue le potentiel inexploité de certaines fonctionnalités offertes par la DHT, notamment l'absence de stockage des données de la blockchain de manière répartie sur les pairs du réseau. En effet, les développeurs d'Ethereum ont fait le choix d'implanter une DHT, dérivée de Kademlia (confère le chapitre 1), pour son réseau P2P, mais son utilisation se limite à la découverte de nouveaux nœuds du réseau à travers le protocole *Node Discovery*. Or, la taille du stockage totale d'un fullnode nouvellement synchronisé s'élève à environ 600 Go, ce qui représente 18 pétaoctets à l'échelle du réseau actuel composé d'environ 30 000 nœuds. De plus, la blockchain, par définition, croît continuellement et par extension la taille des données stockées également. Ce scénario n'est pas tenable dans le temps. En particulier, le cap des 1 To de stockage sera bientôt atteint au risque d'engendrer une diminution du nombre de nœuds qui ne seront pas tous en mesure d'accroître leur espace de stockage au delà de cette limite.

Dans ce chapitre, nous étudierons tout d'abord comment les données sont stockées actuellement au sein du réseau P2P de manière globale, et plus particulièrement comment cela est géré dans l'implantation du client de référence Geth. Il existe bien sûr d'autres clients logiciels pour exécuter un nœud Ethereum, mais ceux-ci sont soit très minoritaires [KMM+18], soit dépréciés et abandonnés. À titre d'exemple, le client OpenEthereum (précédemment appelé Parity Ethereum), longtemps deuxième client le plus populaire après Geth, est déprécié et son développement a été abandonné en mai 2022. Ensuite, nous proposerons une nouvelle architecture qui

tire pleinement parti de la DHT d'Ethereum pour le stockage efficace des données de manière répartie sur l'ensemble des pairs et nous étudierons sa validité.

5.1 Étude du stockage dans le client de référence Geth

Pour comprendre comment le stockage des données de la blockchain peut être optimisé de manière sûre, nous avons besoin de répondre à trois questions :

— à quelles fins les données sont-elles utilisées dans Ethereum ?
— quelles structures de données sont mises en œuvre ?
— toutes les données sont-elles disponibles/utilisées à tout moment ?

Ethereum est majoritairement mentionné pour l'apport des *smart contracts* à la sphère blockchain. Nous évoquions succinctement leur fonctionnement dans la section 1.3 de l'état de l'art et il s'agit d'une des utilisations principales de la blockchain Ethereum, à côté de l'utilisation en tant que système de paiement décentralisé (pour laquelle Bitcoin reste plus populaire). Ici, nous allons montrer comment ces contrats intelligents sont mis en œuvre à travers les données de la blockchain, à travers quelle architecture et quelles structures de données. Ce travail préliminaire a nécessité l'étude de plusieurs sources notamment la documentation officielle [Them], mais celle-ci a été dépréciée dans le courant de l'année 2020 et déplacée sur le site officiel [Thej]. Cette dernière constitue une bonne base, mais reste lacunaire lorsqu'il s'agit d'étudier en profondeur le fonctionnement des protocoles et implantations. C'est pourquoi nous avons également étudié le code-source et les commentaires de Geth [The22]. Nous avons également utilisé le site web `ethereum.stackexchange.com` et le *yellow paper* Ethereum [Woo14] pour parfaire notre compréhension de l'architecture de stockage actuelle.

5.1.1 Les données et leur utilisation dans Ethereum

De façon abstraite, Ethereum cesse d'être un simple registre distribué et s'apparente à une machine d'états distribuée basée sur l'exécution (ou l'application) de transactions. Cette machine d'états utilise différents types de données pour son fonctionnement, illustrés dans le schéma 5.1 (nous détaillons dans la section suivante les structures de données utilisées et les interactions entre celles-ci).

L'état courant d'Ethereum est appelé *world state* et la mise à jour de cet état est effectuée en appliquant l'ensemble des transactions contenues dans le dernier bloc qui a été créé. Le *world state* est composé de l'ensemble des comptes créés, sachant qu'un compte est l'association d'une adresse d'un utilisateur ou d'un *smart contract* à un état de compte. Dans le cas d'un compte utilisateur, il n'y a pas de code associé donc le hash du code est le hash de la chaine de caractère vide. Pour un compte de *smart contract*, il s'agit du hash du bytecode à exécuter par l'*Ethereum Virtual Machine*, c'est-à-dire l'environnement d'exécution présent sur chacun des nœuds du réseau P2P, pour mettre à jour l'état. Lorsqu'un nouveau bloc est créé, dont les transactions peuvent concerner des échanges de crypto-monnaies ainsi que des appels à des fonctions de *smart contracts*, l'ensemble des nœuds du réseau doit générer le nouveau *world state* et, lorsqu'un consensus au niveau des blocs est atteint, tous les nœuds partagent alors le même état. C'est pour cela qu'Ethereum est qualifié d'« ordinateur unique et conforme ».

À côté du *world state*, il y a également le stockage des transactions récupérées à partir des blocs de la blockchain. Parmi les éléments importants d'une transaction figurent le prix en gaz pour l'exécuter, l'adresse du destinataire (qui peut être un smart contract) et la valeur de la transaction. Enfin, un objet intermédiaire est généré et manipulé par les clients, appelé un « reçu » (*receipt* en anglais). Il s'agit d'un objet qui représente le résultat d'une transaction

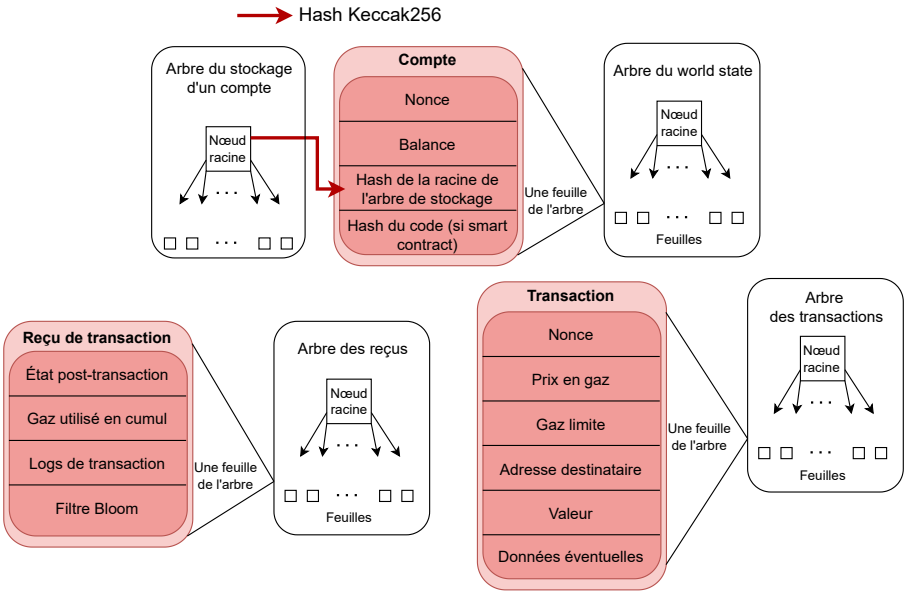

FIGURE 5.1 – Illustration des différents types de données dans Ethereum.

encapsulée dans un bloc, comprenant une empreinte numérique de la transaction, son numéro de bloc, la quantité de gaz utilisée et, en cas de déploiement d'un contrat intelligent, l'adresse de celui-ci.

5.1.2 Structures et bases de données utilisées

La brique de base des structures de données utilisées dans Geth est un arbre combinant deux autres types d'arbres : l'arbre de Merkle et l'arbre PATRICIA (*Practical Algorithm to Retrieve Information Coded in Alphanumeric*). Cet arbre fournit une structure de données de type clé-valeur basé sur des préfixes communs (arbre PATRICIA) et permet de vérifier facilement l'intégrité en comparant le hash de la racine (arbre de Merkle).

L'arbre de Merkle est une structure de données classique utilisée dans le cadre des systèmes distribués. Sa construction se déroule ainsi : les feuilles de l'arbre sont les hashes de chacune des données initiales, puis ces hashes sont concaténés deux à deux (arbre binaire) pour former le nœud parent et ainsi de suite jusqu'à la racine de l'arbre. Cette racine est alors échangée de manière sûre entre les parties du système distribué pour vérifier efficacement l'intégrité de l'ensemble des données : une seule modification des données entraîne un hash différent de la racine.

L'arbre PATRICIA est un type d'arbre préfixe utilisé pour représenter un tableau associatif. C'est une structure compacte dans laquelle chaque nœud n'ayant qu'un seul enfant est fusionné avec ce dernier. Elle permet d'obtenir une recherche, une insertion et une suppression en $\mathcal{O}(\log_2 N)$ [23], N étant le nombre d'éléments stockés. Le schéma 5.2 illustre un exemple simplifié d'un arbre Merkle-Patricia tel qu'implanté par Ethereum (la taille de la clé est volontairement raccourcie par souci de lisibilité, sa taille est normalement de 32 octets).

L'arbre Merkle-Patricia est la structure de données utilisée pour stocker le *world state*, les

23. tel qu'annoncé par les développeurs de Geth mais dans la pratique, cela est à mettre en perspective avec le fait que le nombre d'éléments de l'arbre peut être bien plus grand face à la taille de clé, k, et la complexité de la comparaison de chaîne de caractère $\mathcal{O}(k)$

transactions dans un bloc, les reçus des transactions ainsi que les données de stockage d'un compte. Dans Geth, cela est mis en pratique dans deux bases de données clé-valeur : LevelDB et FreezerDB. LevelDB est utilisée pour le stockage des éléments (blocs, entêtes, reçus, etc.) dont l'utilisation ou la modification est régulière. Dans la pratique, il s'agit du stockage des éléments obtenus depuis moins de trois *epoch* (pour rappel, une *epoch* correspond à 30 000 blocs, donc 90 000 blocs pour LevelDB). Au-delà de ce seuil, les éléments sont transférés vers la base FreezerDB pour leur stockage à long terme, dans laquelle les accès sont très rares et les modifications impossibles (*append-only*). Dans Geth, LevelDB est aussi appelée *Key-Value Store*, FreezerDB, quant à elle, est aussi appelée *Ancient Store*. Étant donné, les propriétés respectives de ces deux bases de données, on qualifie les données de LevelDB comme *hot data* (stockage prévu sur SSD) et celles de FreezerDB sont qualifiées de *cold data* (stockage prévu sur HDD).

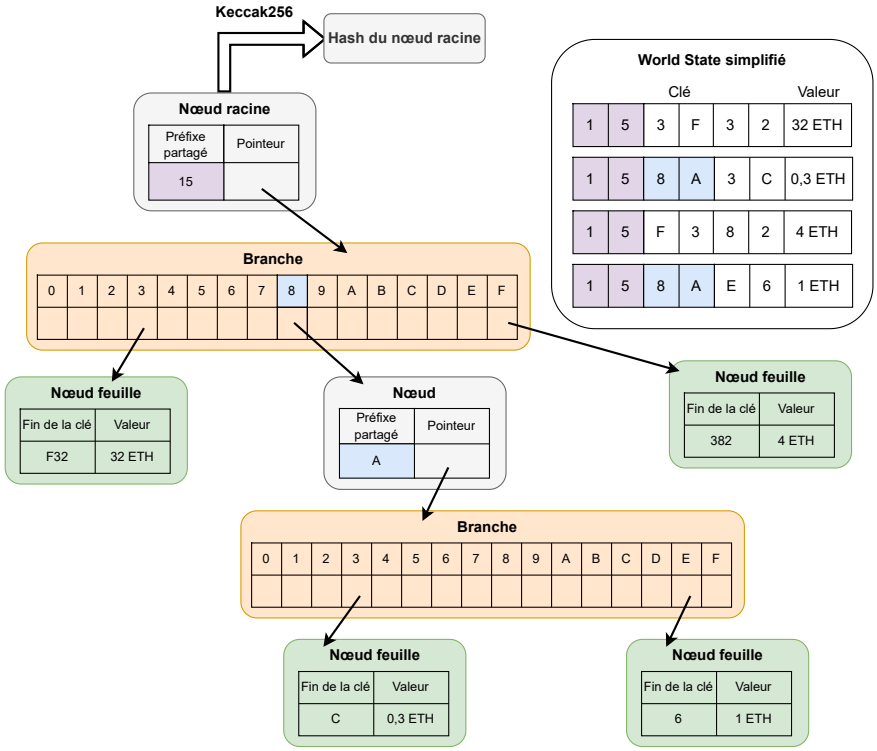

FIGURE 5.2 – Illustration d'un arbre Merkle-Patricia simplifié. Ce schéma est inspiré du travail de Lee Thomas [Tho16].

Dans le graphique 5.3, nous montrons les liens entre ces structures et le contenu d'un bloc. On observe qu'un bloc est composé de trois éléments, à savoir son entête, la liste des transactions qui le composent et la liste des blocs oncles connus (pour rappel, il s'agit des autres blocs au même niveau qui partagent le même parent). L'entête du bloc est composé de nombreux éléments que nous ne détaillerons pas tous ici. MixHash et Nonce constituent la preuve de travail (on peut vérifier que le crypto-puzzle est bien résolu). La limite de gaz autorisé et le total de gaz utilisé dans ce bloc sont également stockés dans l'entête. Les hashes des racines des arbres Merkle-Patricia des comptes, des transactions et reçus contenus dans le bloc sont ajoutés dans l'entête afin qu'un pair du réseau recevant ce bloc puisse vérifier que lorsqu'il applique les transactions sur son propre *world state*, il obtient le même état.

Cet arbre Merkle-Patricia, malgré sa structure optimisée, montre ses limites en termes de per-

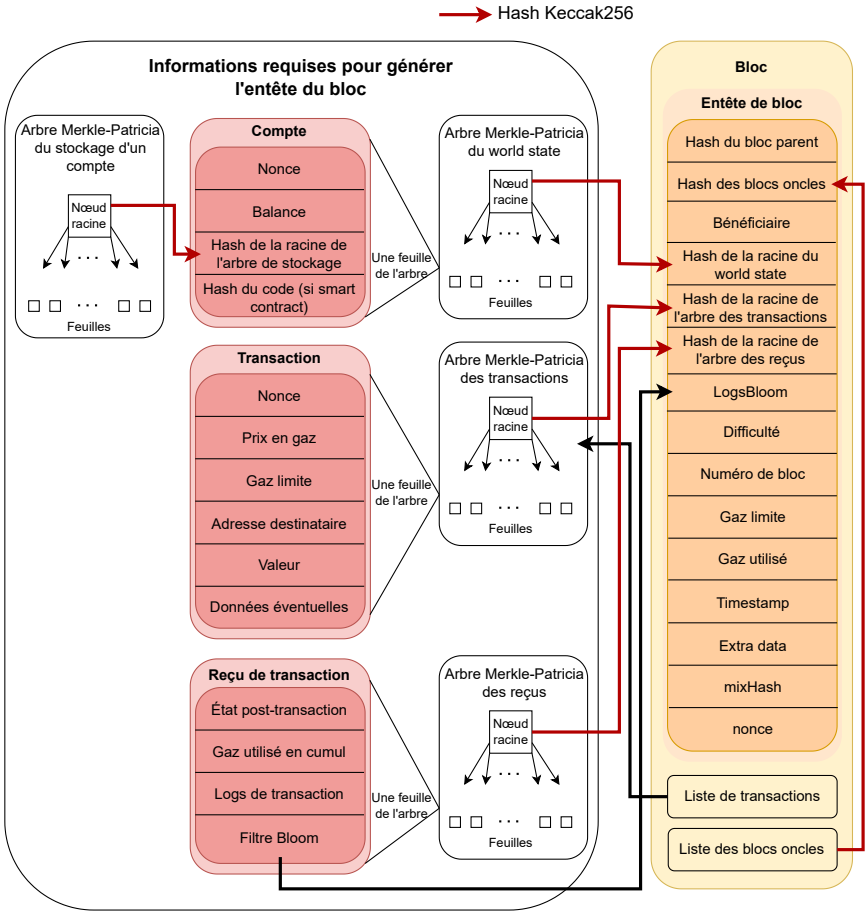

FIGURE 5.3 – Illustration du contenu d'un bloc et des liens avec les arbres du world state. Plusieurs éléments, comme la signature ECDSA, ont été omis pour préserver la lisibilité. Ce schéma est inspiré du travail de Lee Thomas [Tho16].

formances d'accès. En effet, la blockchain Ethereum est en constante augmentation, et malgré le stockage des bases de données sur SSD, les nombreux accès aux structures de données nécessaires pour appliquer le contenu d'un bloc commencent à être problématiques pour les performances des clients, notamment lors de la synchronisation [Fou20]. C'est pourquoi les développeurs ont introduit une nouvelle structure de données (à plat) permettant une complexité d'accès en $\mathcal{O}(1)$, appelée Snapshot. Il s'agit de la vue complète de l'état d'Ethereum à un bloc donné de l'epoch courante. De plus, le Snapshot est toujours disponible dans LevelDB, il n'est jamais transféré dans FreezerDB. Nous verrons dans la section suivante les différents modes de synchronisation d'un nœud Geth et notamment son mode par défaut, snap, qui justement tire parti du Snapshot.

5.1.3 Modes de synchronisation dans Geth

Le client Geth propose plusieurs modes de synchronisation : *Snap* (mode par défaut), *Full*, *Light* et un dernier mode plus particulier, *Archive*.

Snap La synchronisation d'un nouveau nœud en mode *snap* se déroule selon cette séquence :
— téléchargement et vérification des entêtes des blocs ;

— téléchargement des corps des blocs et reçus, et en parallèle téléchargement des données brutes du *world state* (feuilles de l'arbre Merkle-Patricia) et reconstruction complète de l'arbre à partir de ces données ;
— correction de l'arbre à la volée en fonction des nouvelles données qui arrivent (*healing phase*).

Un tel nœud conserve également des points de sauvegarde de l'état, mais pas l'intégralité des états. Ce mode permet d'accélérer la synchronisation d'un nouveau nœud, car il récupère directement les états (sans ré-exécuter l'ensemble des transactions) chez les autres pairs. Cette nouvelle méthode de synchronisation, en rupture avec la méthode classique [24], a été introduite avec un mode antérieur, *fast* (depuis la v1.6.0 de Geth en 2017). Ce dernier permettait au pair de partager les états en les récupérant à partir des structures de données historiques, les arbres Merkle-Patricia. Le mode *snap* tire parti de la nouvelle structure Snapshot (accès en $\mathcal{O}(1)$) pour accélérer le partage direct des états, sans avoir à reconstruire ceux-ci depuis les blocs.

Full Le mode *Full* est le mode classique de synchronisation pour une blockchain, il s'agit de récupérer l'ensemble des blocs, de vérifier leur provenance et la preuve de travail et de ré-exécuter les transactions qu'ils contiennent pour mettre à jour *world state* en conséquence. L'intégralité des états intermédiaires depuis le bloc de genèse n'est pas conservée. Régulièrement le nœud procède à un nettoyage des données qui concernent les précédents états et s'il en a besoin, il ré-exécute les transactions entre le point de sauvegarde des états le plus proche et l'état ciblé. En juillet 2022, un tel nœud stockait environ 600 Go de données comme nous le verrons dans le tableau 5.1.

Light Les nœuds légers se synchronisent très rapidement, car ils ne conservent que les entêtes des blocs et ne génèrent pas d'état. Ils peuvent être utilisés pour émettre des transactions, mais ne peuvent pas valider des blocs. Ils dépendent fortement des *fullnodes* qui leur envoient les informations nécessaires de l'état courant (balance des comptes par exemple) pour fonctionner.

Archive Le mode *Archive* fonctionne comme un *fullnode* mais il permet en outre à un nœud de conserver toutes les données et états intermédiaires depuis le bloc de genèse. En septembre 2022, un tel nœud stocke 12 To de données. L'utilité d'un tel nœud n'est pas claire même au sein de la communauté des développeurs d'Ethereum : le cofondateur Vitalik Buterin a annoncé que le réseau P2P Ethereum fonctionne très bien sans nœud d'archive [25].

État du stockage d'un fullnode

Afin de constater la taille du stockage d'un fullnode, nous avons lancé la synchronisation d'un nouveau nœud Geth en mode Snap en juillet 2022. Cette synchronisation a duré environ 26h et la taille du stockage a atteint un peu moins de 600 Go. Geth propose un outil, appelé DB Inspect, pour inspecter les données stockées et générer un résumé que l'on peut voir dans le tableau 5.1. On observe que le stockage des blocs (*bodies*) et des listes des reçus (*receipt lists*) dans FreezerDB occupe environ 60% (358,9 Go) du stockage total (597,14 Go). Comme nous l'avons vu précédemment, ces données n'ont pas vocation à être consultées régulièrement. Il s'agit plutôt de données archivées qui permettent, si besoin, de rejouer l'ensemble des transactions et générer un nouveau *world state* en cas de problème. Cependant, l'utilité de ce stockage est moindre

24. téléchargement des blocs puis exécution des transactions
25. `https://twitter.com/vitalikbuterin/status/1295534697376649217?lang=en`

depuis le changement de paradigme de synchronisation introduit par le mode *fast*, puis *snap*, car FreezerDB est utilisé dorénavant pour accéder aux données "ossifiées" de la blockchain à envoyer à un nouveau fullnode qui se synchronise.

5.2 Nouvelle méthode de synchronisation et de stockage des données

En reprenant les trois questions que nous avons posées précédemment, nous avons vu que les données des blocs étaient utilisées pour la mise à jour du *world state* et qu'au-delà des 90 000 derniers blocs directement accessibles dans LevelDB, elles sont transférées dans la base de données à long terme FreezerDB. Depuis le changement de paradigme de synchronisation, introduit par le mode *fast* (et optimisé par le mode *snap*), les états sont directement téléchargés depuis les autres pairs et non plus générés à partir de l'exécution des transactions contenues dans les blocs récupérés.

Nous avons également remarqué, à travers une expérience illustrée par la Figure 5.4, qu'un nœud par défaut [26] contacte très peu de pairs (entre 250 et 350) et que seule une très faible proportion est réellement utilisée pour récupérer la très grande majorité des blocs. La charge de travail est donc très inégalement répartie.

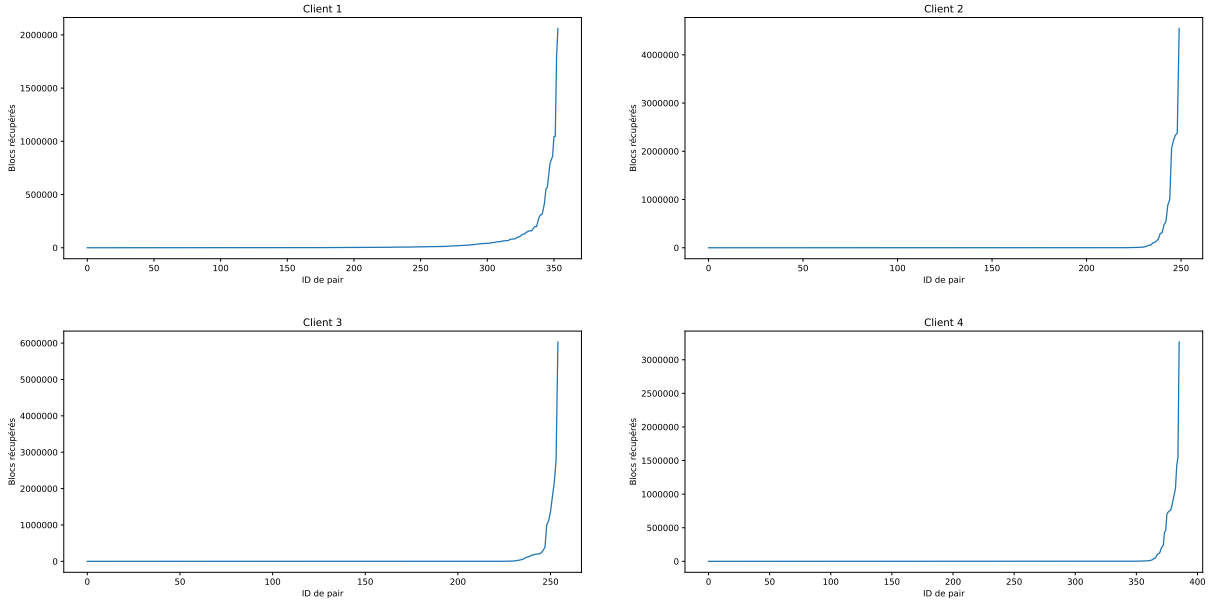

FIGURE 5.4 – Nombre de blocs envoyés par les pairs auquel le nœud qui se synchronise est connecté. Quatre synchronisations indépendantes ont été réalisées.

Partant de ces constats, nous proposons de répartir de manière efficace le stockage de FreezerDB, en particulier les corps des blocs et listes de reçus [27], sur l'ensemble des pairs du réseau grâce à la DHT d'Ethereum. Cela diminuera les besoins en capacité de stockage pour les *fullnodes*. Notre solution ne concerne pas les données présentes dans LevelDB, car celles-ci sont nécessaires pour la génération d'un nouvel état et sujettes à des accès fréquents en lecture et écriture qui

26. mode *snap* et 50 connexions maximum simultanés
27. Dans la suite de ce chapitre, nous parlerons que du stockage des blocs par facilité

TABLE 5.1 – Détail du stockage d'un nœud (Geth) nouvellement synchronisé le 14 juillet 2022 après 26h d'exécution.

Base de données	**Objet**	**Taille**	**Nombre d'objets**
LevelDB	Entêtes de blocs	52,27 Mo	90 002
LevelDB	Corps de blocs	7,41 Go (1,24%)	90 002
LevelDB	Listes de reçus	5,95 Go (1%)	90 002
LevelDB	Codes des smart contracts	3,88 Go (0,65)%	600 111
LevelDB	Nœuds des arbres	130,93 Go (21.93%)	1 148 545 991
LevelDB	Snapshot (compte)	8,66 Go (1,45%)	177 174 087
LevelDB	Snapshot (stockage)	55,25 Go (9,25%)	723 729 535
FreezerDB	Entêtes de blocs	7,22 Go (1,21%)	15 050 309
FreezerDB	Corps de blocs	242,17 Go (40,55%)	15 050 309
FreezerDB	Listes de reçus	116,73 Go (19,55%)	15 050 309
Total		578,25 Go (96,84%)	
Corps de blocs et listes de reçus de FreezerDB		358,9 Go (60,1%)	
Total de LevelDB (prenant en compte les éléments non étudiés ici)		230,21 Go (38,55%)	
Total de FreezerDB (prenant en compte les éléments non étudiés ici)		366,93 Go (61,45%)	
Total (LevelDB + FreezerDB) (prenant en compte les éléments non étudiés ici)		597,14 Go (100%)	

requièrent un stockage local et performant. Comme nous l'avons vu précédemment, un nœud qui souhaite mettre à jour son *world state* a besoin de son précédent *world state* et du dernier bloc connu qui fait consensus et qui n'a pas encore été pris en compte dans le *world state* du nœud. C'est pourquoi nous considérons que chaque pair du réseau a besoin de conserver tous les blocs qui n'ont pas encore été transférés dans FreezerDB.

Pour la répartition du stockage, nous utilisons la fonction de distance XOR, héritée de Kademlia, entre l'ID réseau des pairs (qui pour rappel est l'application de la fonction de hachage Keccak256 à la clé publique ECDSA du pair) et le hash Keccak256 des entêtes des blocs. Ainsi, on peut définir une distance en deçà de laquelle un pair devient responsable de certains blocs, garantissant un certain facteur de réplication, c'est-à-dire un nombre de pairs responsables des mêmes blocs afin de garantir que tous les blocs sont effectivement stockés par plusieurs pairs, rendant le système plus robuste au churn et aux attaques.

5.2.1 Gain de stockage attendu

En adoptant cette nouvelle méthode de synchronisation et de gestion du stockage des données, le réseau P2P Ethereum peut économiser beaucoup d'espace de stockage. Étant donné une taille de stockage des corps de blocs et listes de reçus dans FreezerDB de 358,9 Go (tableau 5.1), le tableau 5.2 montre les gains de stockage attendus à l'échelle d'un pair et à l'échelle du réseau P2P. Pour le calcul du gain sur FreezerDB, nous avons pris en compte aussi le stockage des entêtes des blocs (dans FreezerDB) dont le stockage est fixe sur chaque pair. D'après les données de Crawleth, notre explorateur du réseau P2P d'Ethereum que nous avons présenté au chapitre 4, en septembre 2022 le réseau est composé d'environ 36 000 nœuds. Pour un facteur de réplication de 1125, c'est-à-dire 1125 nœuds responsables du stockage d'un même bloc, le gain du stockage à long terme attendu sur un *fullnode* est d'environ 95% (et environ 58% de gain sur le stockage total du pair). Un tel facteur de réplication est une hypothèse forte assurant une forte fiabilité du stockage. Il serait très difficile de créer assez de nœuds Sybil pour outrepasser 1125 nœuds légitimes et faire disparaître un bloc, si les règles préventives présentées au chapitre 2 sont respectées. Dans la pratique, le taux de réplication pourrait sans doute être abaissé sans danger.

De plus, nous avons également remarqué que la taille du stockage de notre nœud (illustré dans le tableau 5.1), une fois synchronisé, a augmenté plus rapidement que prévu. Les nouvelles données de la blockchain n'expliquaient pas entièrement cette augmentation. D'après Etherscan [28], la tendance de l'augmentation indue du fait de la fuite de données dans les structures est de 50 Go par mois et seule l'exécution d'un nettoyage du nœud (`geth snapshot prune-state`), en hors ligne, permet de corriger cela.

C'est pourquoi notre nouvelle solution de stockage semble appropriée pour pérenniser l'utilisation de *fullnodes* dans Ethereum.

5.2.2 Mise en œuvre dans Geth

Au niveau du protocole de découverte des nœuds, Geth implante deux versions, appelées « Discv4 » et « Discv5 ». En 2022, le protocole « Discv4 » est le seul utilisé, « Discv5 » étant une évolution attendant un déploiement massif. Nous ne détaillerons pas de manière exhaustive les différences entre ces deux versions, mais nous avons dû utiliser « Discv5 » pour notre preuve de concept, car dans « Discv4 » les pairs sont identifiés par leur clé publique alors que dans « Discv5 », ils sont bien identifiés par leur ID réseau (hash Keccak256 de la clé publique), comme il convient de l'être.

28. `https://etherscan.io/chartsync/chaindefault`

TABLE 5.2 – Évaluation théorique du gain en termes de stockage pour un *fullnode*.

Préfixe (bits)	Facteur de réplication	Stockage des blocs et reçus dans FreezerDB par pair (Go)	Gain sur FreezerDB	Gain total par pair	Stockage des blocs et reçus total dans le réseau (Go)
1	18 000	179,45	48,91%	30,05%	6 460 200
2	9 000	~89,73	73,36%	45,08%	3 230 100
3	4 500	~44,87	85,58%	52,59%	1 615 320
4	2 250	~22,44	91,70%	56,35%	807 660
5	1 125	~11,22	94,75%	58,22%	403 830
6	563	~5,61	96,28%	59,16%	201 915
7	282	~2,81	97,05%	59,63%	101 160
8	141	~1,41	97,43%	59,87%	50 580
9	71	~0,71	97,62%	59,98%	25 560
10	36	~0,36	97,71%	60,04%	12 780

Comme évoqué dans la section 5.1.2, Geth utilise deux bases de données, à savoir LevelDB pour l'utilisation normale du client et FreezerDB pour le stockage à long terme. Nous avons modifié les conditions de stockage de cette seconde. Pour cette preuve de concept, nous nous sommes focalisés uniquement sur le stockage des corps de blocs. Le principe reste le même et est aussi applicable aux reçus.

Nous avons ainsi implanté deux nouvelles méthodes de synchronisation au sein du client Geth [Eis22], résumé dans le tableau 5.3. La première option permet de récupérer les blocs souhaités uniquement chez les pairs qui en sont responsables, la deuxième permet de récupérer uniquement les blocs dont le pair est responsable (en utilisant la fonction de distance). Ces deux options peuvent être combinées pour permettre au client de fonctionner comme nous l'avons prévu dans notre nouvelle solution, à savoir récupérer les blocs chez les pairs responsables des blocs que notre client doit stocker de part son identifiant réseau.

TABLE 5.3 – Tableau récapitulatif des nouvelles options de Geth.

Option	Fonctionnalité
`-syncFromDHT`	permet au client de contacter uniquement les pairs responsables d'un bloc souhaité
`-storeInDHT`	permet au client de ne demander que les blocs dont il est responsable du stockage

5.2.3 Évaluation de notre solution

Pour vérifier le bon fonctionnement de notre preuve de concept et évaluer le temps d'exécution et le gain de stockage en conditions réelles, nous avons mené l'expérience suivante. Nous avons déployé 16 nœuds Geth (non modifiés) dans un réseau P2P privé dont les NodeIDs ont été forcés [29] pour obtenir une répartition uniforme sur la DHT afin qu'ils soient chacun responsable de blocs différents. Nous avons également déployé un nœud de type *bootnode* afin que les pairs se découvrent et se connectent les uns aux autres. De plus, nous avons généré environ 7500 blocs pour peupler notre instance de test d'Ethereum, représentant 1,4 Go de données total, que nous avons préchargés sur les 16 clients. L'expérience consiste pour un nœud de test, exécutant notre client Geth modifié avec une combinaison d'options choisie, à rejoindre le réseau des 16 pairs et se synchroniser, c'est-à-dire télécharger les blocs dont il a besoin. Nous avons volontairement choisi un taux de réplication égal à 1 (donc le préfixe commun est configuré à 4 bits) afin que chacun des 16 nœuds soit seul responsable des blocs qu'il stocke. La disposition des 16 nœuds est illustrée dans le schéma 5.5

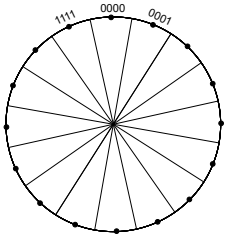

FIGURE 5.5 – Placement forcé (préfixe de 4 bits) des 16 nœuds sur la DHT de notre réseau de test, partitionnant l'espace d'adressage.

Cas d'une synchronisation classique (*full* ou *snap*)

Pour commencer, nous avons voulu évaluer comment se comporte la synchronisation des nœuds en synchronisation classique, c'est-à-dire sans prendre en compte notre nouvelle solution. Le nœud qui rejoint le réseau des 16 pairs se synchronise comme le ferait n'importe quel nouveau nœud rejoignant le réseau P2P Ethereum. Nous observons sur le graphique 5.6 le cumul du nombre de demandes de blocs que le nœud effectue à destination des 16 pairs du réseau P2P. Une requête de demande de blocs peut concerner plus d'un bloc. Sur le graphique 5.7, nous voyons comment les pairs du réseau lui ont envoyé les blocs demandés. On remarque que les demandes de blocs ont été envoyées globalement à l'ensemble des pairs, car chaque pair participe à l'envoi des blocs au client qui se synchronise. Ce comportement n'est pas facilement extrapolable au vrai réseau P2P. Il est probable que seule une petite partie du véritable réseau soit contactée par un nœud cherchant à se synchroniser. On remarque par ailleurs que certains pairs participent plus activement que d'autres. Le nœud 1 par exemple participe nettement moins que les autres. Le temps de synchronisation lors cette expérience s'élève à 27 secondes pour environ 7500 blocs.

29. à l'aide de ce programme que nous avons développé `https://github.com/jpeisenbarth/GenerateEthereumPrivateKeyAroundDHT`

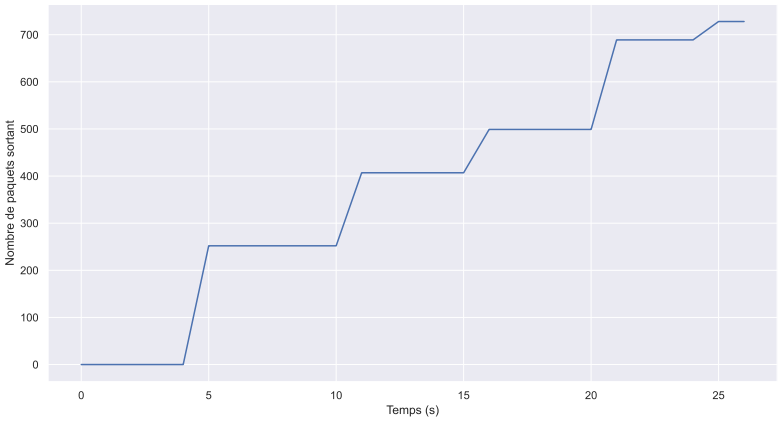

FIGURE 5.6 – Demandes de blocs au cours d'une synchronisation d'un client dans le mode de synchronisation classique rejoignant le réseau.

Cas d'une synchronisation complète par DHT (mode -syncFromDHT)

Notre seconde évaluation concerne une synchronisation classique durant laquelle le nœud qui se synchronise ne demandera les blocs qu'au pair responsable des blocs en question. Ainsi, le nœud évalue avec le hash de l'entête des blocs quel(s) pair(s) (en fonction de son NodeID) contacter pour récupérer un bloc en particulier. Le graphique 5.8 montre que le client demande toujours l'ensemble des blocs, mais sur le graphique 5.9, on observe que, contrairement à précédemment, tous les pairs du réseau contribuent uniformément à l'envoi de ces blocs. Le temps de synchronisation est sensiblement le même que lors de l'expérience précédente.

Cas d'une synchronisation optimisée par DHT (mode -syncFromDHT -storeInDHT)

Notre dernière expérience consiste à tester la validité de notre nouvelle solution de synchronisation. Le nœud qui rejoint le réseau P2P privé ne demandera que les blocs dont il est le responsable. Pour rappel, cela signifie que ce sont les blocs dont le hash de l'entête est proche, au sens de la notion de distance DHT évoqué à la section 5.2. Le graphique 5.10 montre que le nœud demande beaucoup moins de blocs à télécharger et se synchronise ainsi plus vite[30], à savoir 7 secondes. On observe également sur le graphique 5.11 qu'un seul pair est sollicité, car il est le seul responsable des blocs demandés, ce qui est dû à notre configuration du taux de réplication à 1. Dans cette configuration, le nœud qui se synchronise profite d'un gain de stockage par rapport aux modes de synchronisation précédents.

Le tableau 5.4 résume les caractéristiques que nous avons observées lors de notre évaluation. Nous avons également calculé l'écart-type du nombre de paquets de bloc téléchargé par le client qui se synchronise et on observe bien qu'avec l'utilisation de la DHT, la répartition du travail est bien plus homogène (comme le montre la valeur de l'écart type). Dans le cadre de la synchronisation optimisée par DHT, le calcul de l'écart-type ne s'applique pas vraiment, car dans notre expérience, il n'y a qu'un seul nœud qui envoie les paquets au nœud qui se synchronise.

30. Dans le cadre d'une synchronisation de la véritable blockchain, le nœud aurait également le Snapshot à télécharger.

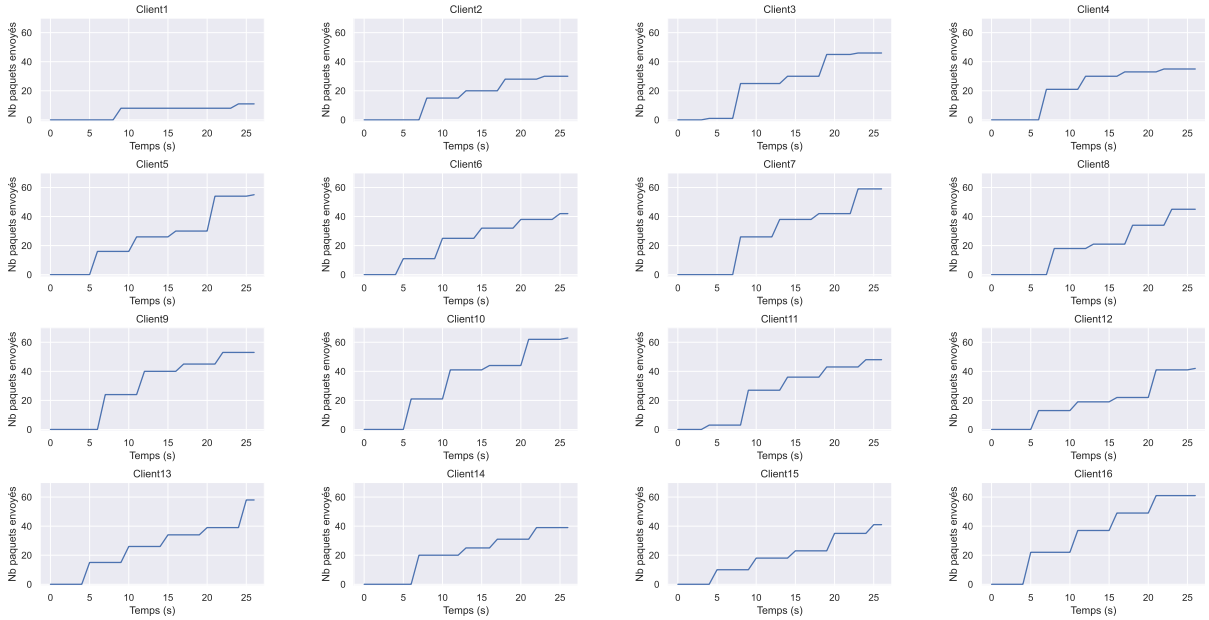

FIGURE 5.7 – Envois des blocs par les clients du réseau (synchronisation classique).

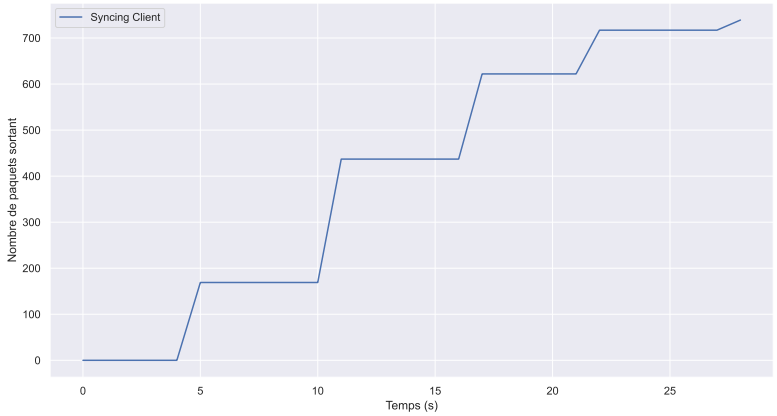

FIGURE 5.8 – Demandes de blocs au cours d'une synchronisation d'un client dans le mode synchronisation complète par DHT rejoignant le réseau.

Conclusion

Dans ce chapitre, nous avons montré comment il est possible d'utiliser l'implantation de la DHT d'Ethereum pour stocker les données rarement utilisées de la blockchain qui sont actuellement sauvegardées dans FreezerDB pour Geth. Plus spécifiquement, nous avons montré qu'en optant pour une stratégie de stockage répartie au sein du réseau, il est possible de réduire considérablement la taille du stockage à long terme pour chaque pair, de l'ordre de 95% du volume de FreezerDB et 58% du volume total, pour un facteur de réplication de 1125 pairs. À l'échelle de l'ensemble du réseau P2P, en estimant à 36 000 le nombre de *fullnodes* devant stocker à l'heure actuelle environ 367Go de données à long terme, cela représente un gain global d'environ 12,5 pétaoctet.

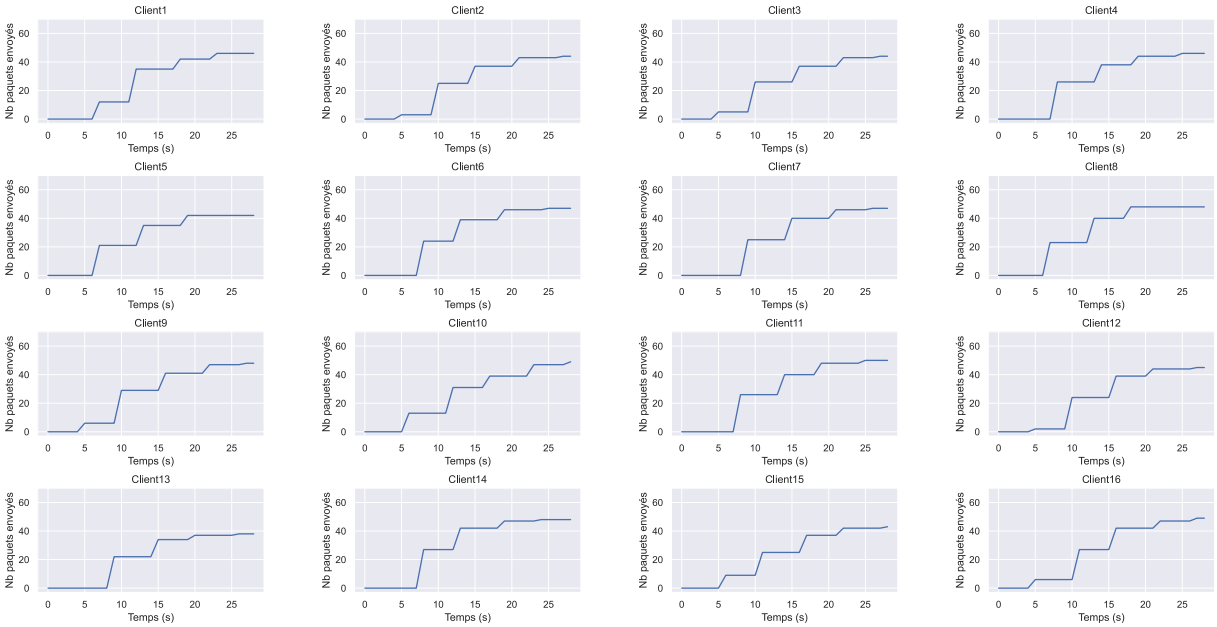

FIGURE 5.9 – Envois des blocs par les clients du réseau (synchronisation complète par DHT).

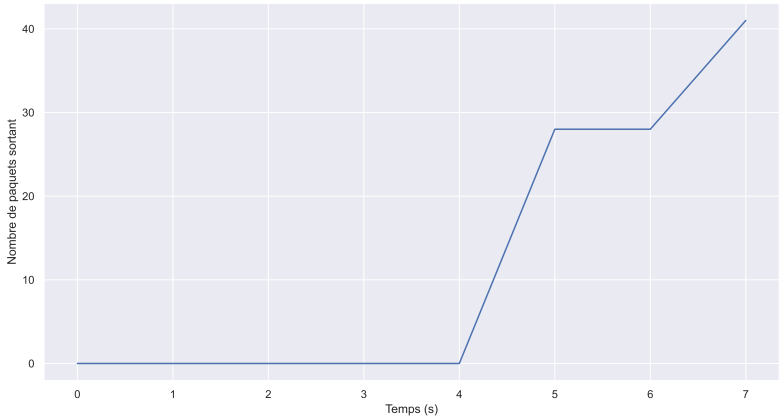

FIGURE 5.10 – Demandes de blocs au cours d'une synchronisation d'un client dans le mode de synchronisation optimisée par DHT.

L'évaluation de notre preuve de concept montre que cette nouvelle méthode de stockage et de synchronisation des blocs fonctionne très bien et il est tout à fait envisageable pour la communauté Ethereum de développer et adopter une telle stratégie de synchronisation et gestion du stockage des blocs afin de palier le problème de stockage rencontré par les administrateurs des *fullnodes*. Notre nouvelle stratégie de synchronisation peut être déployée progressivement, chaque client pouvant choisir sa méthode de synchronisation, toutes étant compatibles. Cette stratégie n'affecte également aucun mécanisme au cœur de la blockchain Ethereum. Il est toujours possible pour un nœud de récupérer l'intégralité des données, même si cette méthode de synchronisation perd grandement de son intérêt. Notre solution est également configurable au niveau du taux de réplication qui permet de trouver un compromis entre le gain en stockage et le risque de perte de données (lié au churn par exemple).

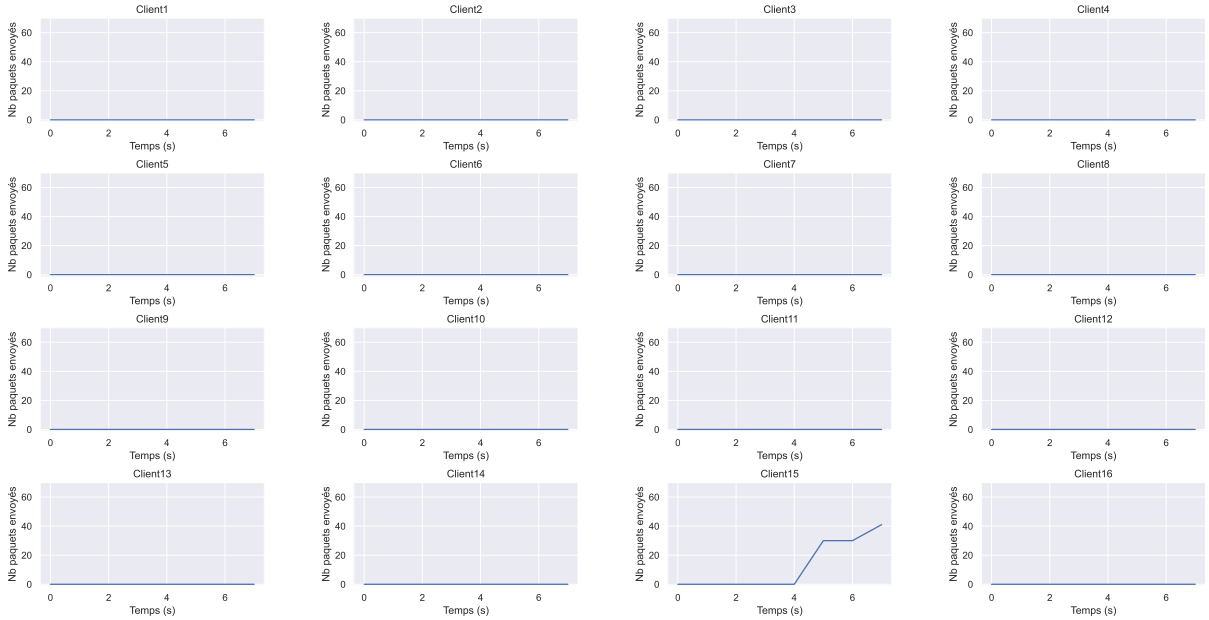

FIGURE 5.11 – Envois des blocs par les clients par défaut du réseau (synchronisation optimisée par DHT).

TABLE 5.4 – Tableau récapitulatif de l'évaluation des nouvelles options de Geth.

Type de synchronisation	Option	Temps de synchronisation	Écart type du nombre de paquets de blocs envoyés par pair
Classique	∅	27 s	∼12.91
Complète par DHT	`-syncFromDHT`	27 s	∼3.02
Optimisée par DHT	`-syncFromDHT` `-storeInDHT`	7 s	Non applicable

Cependant, l'utilisation de la DHT pour le stockage efficace des données fait d'Ethereum une cible de choix d'attaques Sybils permettant de contrôler les données stockées dans la DHT et potentiellement de mener des attaques de déni de service ciblées, telles qu'évoquées dans le chapitre 2. C'est pourquoi il est primordial de s'assurer qu'une telle attaque ne puisse pas avoir lieu. Dans le chapitre suivant, nous étudierons diverses métriques pour quantifier la vraisemblance d'une attaque Sybil sur Ethereum et dans le chapitre d'après, nous proposerons un système de supervision et révocation de pairs ayant des comportements suspects de type Sybil afin de prémunir le réseau de telles attaques.

Chapitre 6

Symptômes d'attaques Sybil contre la DHT d'Ethereum

Sommaire

Introduction

Nous avons décrit et mis en pratique, dans le chapitre 5 de ce manuscrit, un mécanisme de stockage de la blockchain Ethereum utilisant la force de la DHT pour répartir la charge du stockage sur l'ensemble des pairs du réseau.

Mais comme tous les réseaux P2P publics entièrement distribués, c'est-à-dire sans autorité centrale qui contrôle l'accès au réseau, le réseau P2P d'Ethereum est vulnérable aux attaques Sybil, comme nous l'avons vu lorsque que nous avons décrit cette attaque dans l'état de l'art, section 2.1.1. Or, la fiabilité des réseaux P2P repose principalement sur la bonne répartition de la charge de travail sur de nombreux pairs indépendants. C'est pourquoi tout groupe de nœuds qui semble altérer la répartition homogène de la charge de travail entre les pairs peut être dangereux, qu'il s'agisse d'une utilisation anormale du réseau, d'une mauvaise configuration ou d'une attaque intentionnelle de type Sybil. Un attaquant pourrait facilement affaiblir le réseau P2P en créant un grand nombre d'identités à un coût presque nul pour obtenir une influence disproportionnée dans le réseau.

Quelle que soit la cause fondamentale, notre objectif dans ce chapitre est de faire l'inventaire des ensembles de nœuds qui semblent concentrer trop de poids dans le réseau et qui peuvent donc être considérés comme suspects. Dans le prochain chapitre 7, nous présenterons une architecture prévenant les attaques sybils, capable de non seulement de détecter les nœuds suspects à l'échelle du réseau P2P mais ensuite d'annoncer ces nœuds publiquement sur la blockchain via un *smart contract* et d'effectuer leur révocation de manière entièrement distribuée du côté client.

6.1 Modèle d'attaquant et scénarios

Dans notre modèle, nous supposons que tous les participants, y compris les attaquants, sont capables de créer de nouveaux identifiants de nœuds (en générant une paire de clés) à volonté. Ces nouvelles identités ne coûtent rien et ne peuvent pas être liées aux identités précédentes.

Un adversaire peut falsifier plusieurs identités dans le réseau pour mener une attaque Sybil. Cette attaque peut entraîner des retards dans la propagation des blocs ou des transactions et ainsi favoriser un pool de mineurs contrôlé par l'adversaire [NG17]. Elle peut également être exploitée pour mener des attaques (D)DoS, des attaques de majorité [ZL19], des attaques du protocole de mixing [BOLL14], et ainsi avoir un impact encore plus important.

En surveillant les nœuds du réseau, notre système pourrait détecter les pairs suspects qui constituent une menace d'attaque Sybil et faire circuler l'information aux pairs du réseau.

L'adversaire peut également partitionner le réseau (via une attaque Eclipse, confère section 2.1.1) pour réduire le coût et maximiser le gain de cette attaque [HKZG15, MHG18]. Si notre nœud de surveillance est inclus dans la partition, il détectera facilement la partition en surveillant la difficulté des nouveaux blocs [NEIP18] : la difficulté diminuerait fortement au fil du temps dans ce cas.

Nous considérons trois catégories de nœuds suspects qui pourraient être liés parce qu'ils partagent une adresse IP commune, un même sous-réseau ou des identifiants de nœuds très proches. Plus précisément, nous voulons identifier dans nos jeux de données les caractéristiques suivantes :

— un sous-réseau contenant plus de nœuds qu'un seuil défini (10% de sa taille par défaut) ;
— des identifiants de nœuds statistiquement trop proches[31] dans la table de hachage distribuée par rapport à la distribution uniforme théorique ;
— une adresse IP responsable de plus d'une identité (Node IDs définis par les clés publiques).

6.2 Jeu de données

La détection des nœuds suspects se base sur un jeu de données qui est généré à partir de celui que Crawleth produit (confère chapitre 4, section 4.1.5). Il s'agit d'un jeu de données intermédiaire permettant de faciliter l'implantation de notre système de détection. Il se présente en deux parties distinctes et reprend la même temporalité que le jeu de données sur lequel il est basé, à savoir une première partie qui concerne le mois de septembre 2021 et une seconde partie qui débute le 15 février 2022 et s'étend jusqu'au 7 mars 2022. Chaque jeu de données est découpé en trois fichiers (dont un extrait est présenté en annexe C) qui reprennent les critères de détection que nous avons établis :

— « subnets.json » contient l'ensemble des pairs dont le nombre d'adresses IP dans un même sous-réseau dépasse un certain seuil ;
— « groups.json » contient les pairs dont l'identifiant de nœuds semble incorrectement reparti dans l'espace d'adressage de la DHT ;
— « aliases.json » contient les pairs qui possèdent un nombre d'identifiants qui dépasse un certain seuil.

Ces critères seront détaillés par la suite.

6.3 Types de symptômes évoquant une attaque Sybil

31. Nous détaillons cette notion dans la section 6.3.2

6.3.1 Répartition des pairs par sous-réseau

Pour commencer, un premier type de comportement suspect est la concentration de pairs, a priori indépendants, au sein d'un même sous-réseau. Nous utilisons la notation *Classless Inter-Domain Routing* (CIDR) pour définir les sous-réseaux. Plus particulièrement, nous considérons des sous-réseaux dont le masque de sous-réseau est, en notation CIDR, /24. En IPv4, cela correspond à des sous-réseaux de taille 256 adresses auxquelles on peut soustraire 2 adresses (adresse de broadcast et adresse de réseau), ce qui donne 254 adresses utiles.

Nous avons défini un seuil arbitraire de 10% de la taille maximale du réseau au-delà duquel nous considérons que le sous-réseau concentre trop de pairs (représentés par leurs adresses IP) hébergeant un nœud Ethereum. En pratique, nous partons du principe que trouver plus de 25 nœuds dans un seul sous-réseau est assez suspect. Sur l'ensemble du mois de septembre 2021, nous avons trouvé cinq sous-réseaux /24 contenant un total de 214 nœuds uniques. En moyenne, notre système de détection a pu trouver 64 nœuds par heure appartenant à ces quelques sous-réseaux surpeuplés.

Concernant la période de février–mars 2022, nous avons trouvé sept sous-réseaux contenant un total de 253 nœuds uniques et en moyenne notre système a trouvé environ 70 nœuds par heure dans ces sous-réseaux.

Nous en avons conclus que ces petits nombres ne sont pas particulièrement inquiétants pour la bonne santé du réseau P2P.

6.3.2 Distribution des identifiants des pairs sur la DHT

Le deuxième et troisième comportement suspect concernent plus particulièrement l'utilisation de la table de hachage distribuée. En effet, un attaquant peut réaliser une attaque Sybil plus subtile en plaçant intentionnellement quelques pairs à proximité d'une identité cible dans la DHT afin d'attirer les requêtes vers lui. On parle alors de la densité des pairs sur la DHT.

Pour commencer l'analyse, nous pouvons d'abord observer si la distribution des pairs sur la DHT est bien uniforme. Pour l'ensemble du mois de septembre 2021, on peut voir sur le graphique 6.1 que la distribution de la taille du préfixe de l'identifiant de nœud partagé entre deux voisins directs dans la DHT est bien répartie, malgré quelques irrégularités à la fin de la courbe qui ne sont pas très significatives et explicables par le fait que trop peu de pairs sont trouvables à ce niveau de proximité pour représenter parfaitement la distribution.

Pour confirmer cette analyse, nous pouvons comparer la distribution mesurée à celle théorique. Pour cela, nous définissons la fonction F donnant le nombre moyen de pairs du réseau partageant x bits avec un autre pair étant donné le nombre total N de pairs dans le réseau. Notre outil a pu trouver 439 561 identifiants de nœuds uniques, donc $N = 439561$ ici.

$$F(x) = \frac{N}{2^x} \tag{6.1}$$

Le tableau 6.3.2 présente quelques valeurs remarquables pour $N = 439561$ et $x \in [1; 256]$. De plus, le préfixe moyen partagé entre deux pairs consécutifs est donné par $prefix_{moy} = \log_2(N) = 18,75\ bits$.

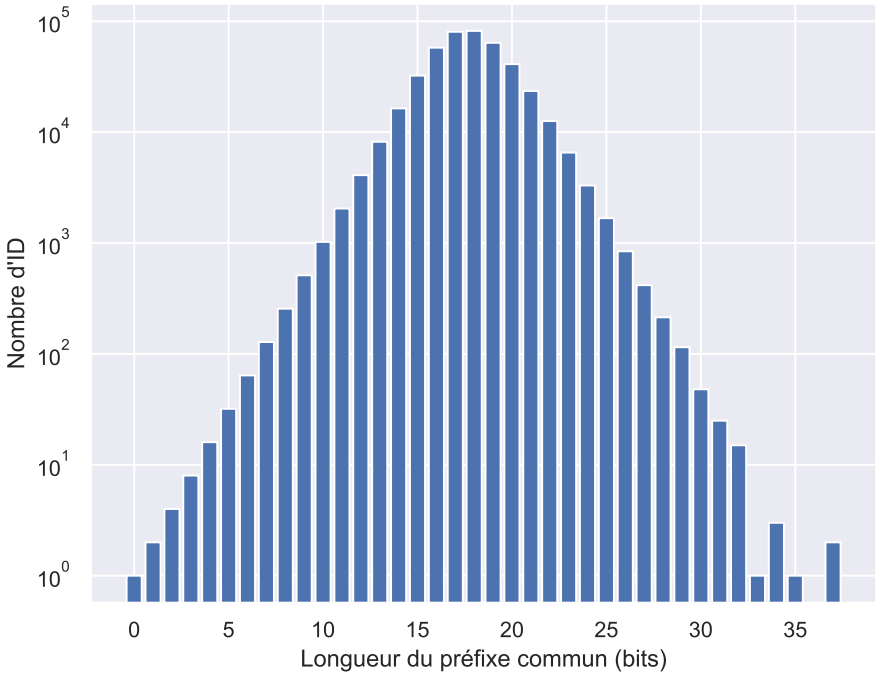

FIGURE 6.1 – Distribution du partage d'un préfixe commun entre voisin sur la DHT

Nombre de bits en commun	Nombre moyen de pairs totaux
1	219 780,5
4	27 472,56
8	1 717,03
12	107,31
16	6,71
20	0,41
24	0,03
28	$1{,}64 \times 10^{-3}$
32	$1{,}02 \times 10^{-4}$
64	$2{,}38 \times 10^{-14}$

Nous montrons dans le graphique 6.2 l'écart entre les données mesurées et théoriques, représentées par l'équation 6.1. Nous pouvons voir qu'il n'y a pas de déviation évidente entre les voisins dans la DHT. Plus particulièrement, concernant la déviation que l'on observe pour un préfixe de 37 bits communs, en pratique on trouve 4 pairs partageant un même préfixe de 37 bits ce qui n'est pas significativement suspect. Nous avons observé la même tendance dans notre jeu de données de février 2022.

Néanmoins, il est possible que quelques attaques localisées puissent se produire, mais sans être visibles dans les statistiques calculées au niveau du réseau. Typiquement, un groupe de 5 pairs partageant un préfixe commun de 22 bits serait statistiquement improbable sans affecter l'ensemble de la distribution. Au moins, nous pouvons conclure qu'aucune attaque très localisée

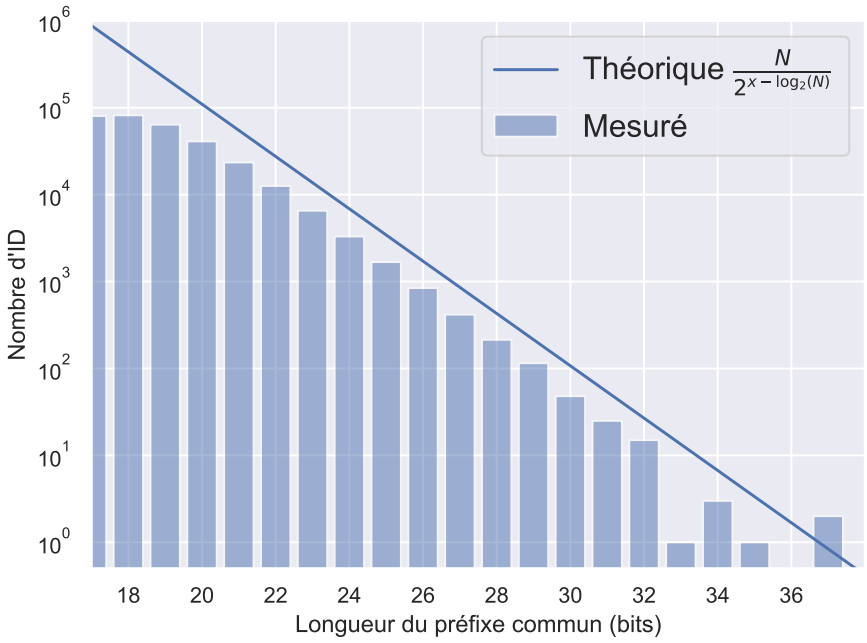

FIGURE 6.2 – Nombre de pairs théorique et mesuré partageant un préfixe commun sur la DHT

(c'est-à-dire des voisins partageant un préfixe très élevé) n'a lieu, ni aucune attaque localisée massive qui fausserait la distribution. Cela peut s'expliquer par le fait qu'aucune donnée n'est aujourd'hui réellement stockée en tirant parti de la DHT d'Ethereum, de sorte que, de telles attaques n'ont actuellement aucun intérêt (notre preuve de concept du chapitre 5 n'a pas été déployé sur le réseau Mainnet).

6.3.3 Concentration d'identifiants par pair

Nous avons remarqué qu'il y a des nœuds qui détiennent de nombreuses identités de nœuds (c'est-à-dire des clés publiques). Nous avons décidé de surveiller les nœuds qui détiennent plus de deux identités. Pour la période de crawl en septembre 2021, cette surveillance a révélé 7 780 adresses IP uniques détenant 396 963 ID de nœuds uniques. Les nœuds qui ont forgé le plus grand nombre d'identités détenaient plus de 10 000 identifiants de nœuds (maximum de ∼14 000). En moyenne, notre robot d'exploration a pu trouver 1 217 nœuds (21 942 identifiants de nœuds) par heure correspondant à ce critère.

Pendant le pic de nœuds en février 2022, ce comportement a été exacerbé : 9 500 adresses IP uniques détenaient 183 731 identifiants de nœuds uniques et 13 % des adresses IP détenaient 75 % des identifiants de nœuds. 69,2 % de ces alias sont détenus par les 10 premiers AS qui hébergent la plupart des adresses IP du réseau (voir la section 4.2.2) et sont situés aux États-Unis ou en Europe. À titre de comparaison, le crawler a pu détecter 2 486 nœuds déviants par heure pendant cette période.

La concentration d'alias dans un très petit nombre d'adresses IP est un signe typique du scénario d'attaque Sybil où un attaquant forge de nombreuses identités et souhaite en tirer profit. Ces mesures montrent que le réseau P2P Ethereum est bien vulnérable à l'insertion de Sybils.

Les conséquences de ce type d'attaque peuvent être diverses. À l'heure actuelle, il est pos-

sible que la concentration d'identifiants altère le bon fonctionnement des protocoles de mixing [BOLL14] voire fournisse un levier de dé-anonymisation des pairs participant à ces protocoles outre la surveillance des échanges de crypto-monnaies. De plus, avec notre architecture de stockage de la blockchain réparti sur l'ensemble des pairs du réseau, présenté à la partie 5, ces attaques peuvent avoir un impact encore plus important. En effet, un contrôle de la DHT permet également d'éclipser certaines données ou d'effectuer des attaques par déni de service sur certains pairs responsables de données possiblement critiques.

6.3.4 Supervision en temps réel

Afin de suivre ces indicateurs en temps réel, nous avons développé un site web regroupant des vues graphiques sur les 3 catégories de nœuds suspects. L'image 6.3 montre la page principale de ce site web. Nous avons également déployé une API REST permettant de récupérer des informations sur les pairs détectés. Par exemple, l'URL `http://crawleth.loria.fr:5000/api/latest/deviant/aliases` permet de télécharger la liste des pairs du dernier crawl étiquetés comme suspects du point de vue de la concentration d'identifiants par pair. L'URL `http://crawleth.loria.fr:5000/api` permet de voir la liste des requêtes pouvant être faites sur l'API REST. Cette API est utilisée par notre outil d'annonce et révocation des nœuds Sybils que nous présentons dans le chapitre suivant.

FIGURE 6.3 – Site WEB de présentation de notre solution de supervision de comportements suspects sur Ethereum

Conclusion

En conclusion, d'une part, la DHT d'Ethereum montre des chiffres rassurants comme une quantité négligeable de sous-réseaux /24 contenant de nombreux pairs et aucun placement intentionnel évident de pairs dénoté par une forte concentration sur l'espace d'adressage. Cependant,

certains nœuds détiennent plusieurs identités dans le réseau, jusqu'à des milliers pour les plus grands. Par ailleurs, nous rappelons que nous avons mesuré une augmentation soudaine et importante du nombre de pairs durant la période février–mars 2022. Fort heureusement, il s'avère que ces nouveaux arrivants n'ont pas eu d'impact sur les trois métriques signalant les nœuds suspects. Néanmoins, ce type d'événement massif combiné à la possibilité pour un nœud de créer de nombreuses identités plaident pour une surveillance permanente du réseau P2P et des mécanismes de prévention des attaques Sybil à grande échelle. Dans le chapitre suivant, nous allons présenter un système, que nous avons développé, permettant de diffuser l'identité de pairs ayant des comportements suspects afin de prévenir leurs potentiels agissements malveillants à travers un mécanisme de révocation.

Chapitre 7

Prévention d'attaques Sybil sur la DHT Ethereum

Sommaire

Introduction

Nous avons vu dans le chapitre précédent que l'implémentation de la DHT d'Ethereum (héritée de Kademlia) est perfectible en termes de protections contre les attaques distribuées ciblant les identifiants de l'espace d'adressage. Ces attaques, si elles sont avérées, peuvent affecter la sécurité de la blockchain, comme nous l'avons précédemment montré. De plus, cette faiblesse empêche l'utilisation de la DHT dont nous avons montré une utilité majeure dans la distribution du stockage au chapitre 5.

Pour toutes ces raisons, il est devenu nécessaire de penser et développer des protections de la DHT Ethereum contre les attaques Sybil. Pour cela, deux solutions existent : contraindre la génération des identifiants ou alors détecter de potentiels attaquants et prévenir les pairs du réseau qui peuvent alors appliquer des règles de révocation à l'encontre de ces pairs suspects. Dans ce chapitre, nous allons présenter notre système de prévention d'attaque Sybil qui appartient à cette deuxième catégorie de solutions. Celui-ci consiste à utiliser les données de supervision de Crawleth pour avoir une vue globale de la DHT et, via un *smart contract*, de publier la liste des nœuds considérés comme suspects. Les pairs du réseau peuvent alors, après vérification, révoquer les connexions à ces nœuds en autonomie.

7.1 Sybil-Prevention : un système distribué de diffusion de nœuds suspects

7.1.1 Architecture

Notre architecture de prévention des attaques Sybil est composée de trois parties, comme le montre le diagramme 7.1. La première partie (à gauche) consiste à surveiller l'ensemble du réseau P2P Ethereum et à détecter les nœuds suspects. La surveillance est effectuée par un ou plusieurs explorateurs validés qui peuvent exécuter notre outil libre Crawleth ou toute autre implémentation de crawler Ethereum. Ceci est nécessaire pour obtenir une vue complète du réseau. Une surveillance entièrement distribuée effectuée uniquement par les pairs sur la base de leur table de routage est limitée à une vue partielle qui n'est précise que localement autour du pair. Cela n'est pas suffisant pour détecter une attaque se propageant sur l'ensemble de l'espace d'adressage de la DHT.

Une fois qu'un crawl est effectué, les informations sur les nœuds qui portent de multiples identités sont disponibles. La détection des nœuds suspects se fait en appliquant les seuils définissant les trois catégories présentées dans la section 6.3. La deuxième partie (au centre) est la diffusion de l'information de ces nœuds à tous les pairs du réseau grâce à notre *smart contract* détaillé plus loin dans cette section. Après son exécution, tous les nœuds du réseau qui écoutent ces événements sont notifiés, par les journaux d'événements, des nouveaux nœuds suspects.

La troisième et dernière partie (à droite) de notre architecture prévenant les attaques Sybil implique chaque nœud du réseau P2P. Ils doivent vérifier les informations reçues par le biais de quelques requêtes FINDNODE ciblées afin d'être témoins de la menace détectée par le crawler. Cette étape de vérification est essentielle pour maintenir pleinement la nature distribuée du réseau P2P et atténuer le rôle du crawler comme simple pointeur vers les nœuds suspects sans aucune autorité sur les pairs qui restent totalement autonomes. Une fois vérifié, chaque nœud révoque les nœuds suspects en coupant les connexions possibles qu'il a avec eux et empêche les futures connexions en constituant une liste noire. Cette dernière partie peut être intégrée directement dans le client ou utiliser un outil externe que nous avons développé et qui est présenté dans la Section 7.2.

Les travaux précédents proposant des mécanismes de défense contre les attaques Sybil en général étaient soit centralisés en s'appuyant sur une autorité centrale pour valider les pairs rejoignant le réseau [DH06], soit entièrement distribués, basés sur une détection/révocation locale des pairs suspects [CCFD13] mais manquant d'une vue exhaustive de la DHT pour corréler les Sybils présents à différents endroits. L'originalité de notre approche est de concilier le meilleur des deux mondes : une vue globale et précise du réseau réalisée par un crawler associée à un mécanisme de révocation entièrement distribué afin de limiter la confiance accordée au crawler et éviter tout risque de détournement et d'abus. Nous nous appuyons sur la diffusion efficace de blocs à tous les pairs pour partager la connaissance des pairs suspects à l'ensemble du réseau, ce qui n'était pas facilement réalisable dans les systèmes historiques utilisant des DHTs qui ne bénéficiaient pas d'une seconde topologie de diffusion.

7.1.2 Smart Contract

Nous avons conçu le *smart contract* présenté dans le Listing 1 pour utiliser les journaux d'événements d'Ethereum afin de distribuer les informations sur les nœuds suspects. En effet, dans la blockchain Ethereum, chaque bloc possède un champ `logsBloom` qui permet à tout utilisateur de rechercher un journal de transaction spécifique (ainsi que les champs indexés du journal). Ces

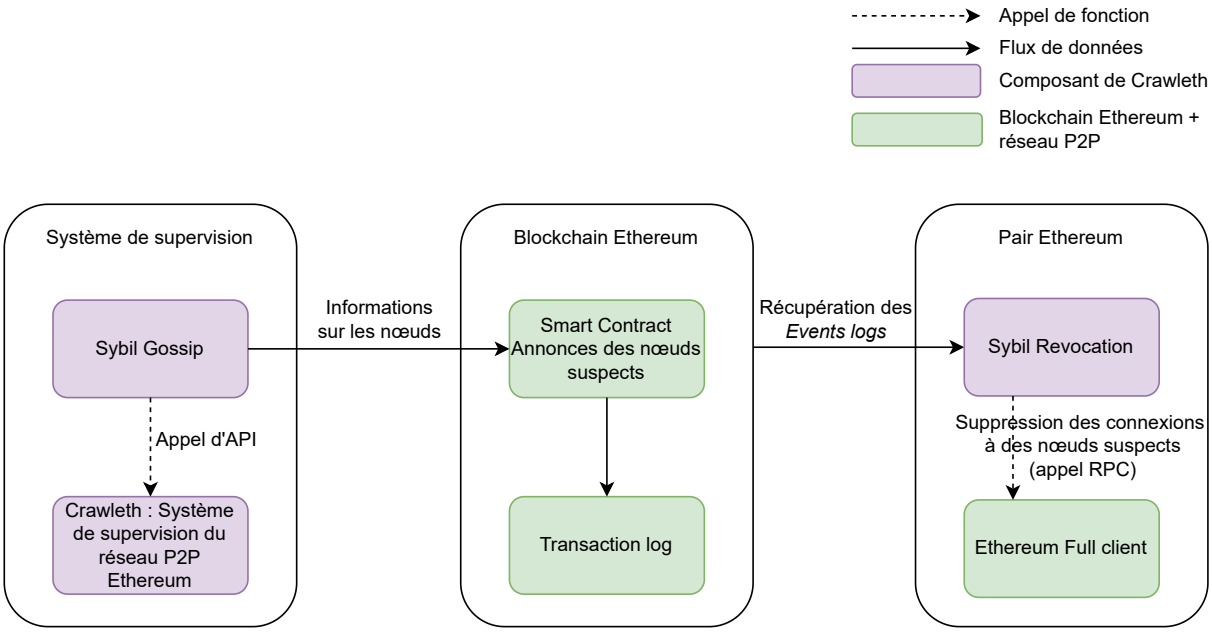

FIGURE 7.1 – Architecture du système de supervision et révocation de nœuds suspectés de participer à une attaque Sybil

journaux sont publiquement accessibles et peuvent être générés à nouveau à partir de la transaction qui les contient. L'événement de notre *smart contract* publie l'enode (NodeID@IP :Port) des nœuds suspects mais de manière structurée pour économiser de l'espace dans le contrat intelligent en factorisant les informations communes à un groupe de pairs suspects, et faciliter le processus de vérification. Les données de chaque groupe de pairs suspects sont stockées différemment en fonction de la menace pour laquelle ils ont été identifiés (labellisé comme "Subnet", "Groups" et "Aliases" dans le programme 1). De même, chaque menace a une procédure de vérification spécifique. Alors qu'une seule requête FINDNODE sur le bon ID de nœud peut éventuellement trouver tous les pairs suspects trop proches les uns des autres dans l'espace d'adressage DHT, une requête par nœud suspect doit être effectuée jusqu'à ce que le seuil définissant le comportement suspect soit atteint dans le cas d'un sous-réseau suspect ou d'une adresse IP gérant plusieurs identités. Par exemple, une adresse IP suspecte détenant 100 identités ne nécessite que la vérification de deux d'entre elles par un pair pour permettre la révocation, si le seuil de détection est de deux. Cette procédure de vérification est illustrée dans l'algorithme 1 rédigé en pseudo code. De plus, pour éviter toute surcharge, notre système de surveillance peut étaler dans le temps l'annonce des nœuds suspects dans le *smart contract* si le nombre de nouveaux nœuds découverts dépasse un certain seuil.

Le *smart contract* autorise une liste d'utilisateurs privilégiés (ajoutés par le propriétaire) qui peuvent publier le résultat de différentes instances du crawler. D'autres instances approuvées du crawler peuvent compléter les données si elles sont témoins de l'absence de pairs suspects ou même remplacer l'instance principale si nécessaire pour assurer la continuité du service. Le fait d'avoir plusieurs instances de crawler de confiance contribue à l'exactitude et à la résilience du système de surveillance.

Cependant, le stockage des données dans les journaux d'événements a toujours un coût (en plus du coût de base de la transaction) : 375 *gas* comme coût de base, 375 *gas* supplémentaires par sujets (*topic* en anglais) dans l'événement et 8 *gas* pour chaque octet de données inclus.

Algorithme 1 : Vérification d'une trop forte concentration de NodeID (alias)

```
Data : l, list of suspicious nodes (IPs, ports, nodeids)
Data : t, threshold
nodeid_counter ← atomic counter;
foreach ip ∈ l do
    i ← 0;
    while nodeid_counter ≤ t or i < sizeof(nodeids) do
        if verify_existence(ip,port,nodeid) then
            nodeid_counter++;
        end
        i++;
    end
    if nodeid_counter > t then
        revoke(ip);
    end
end
```

Pour un événement non anonyme, le premier sujet est la signature de l'événement (nom et type des arguments). Tous les arguments indexés sont traités comme des sujets supplémentaires. Les sujets sont utilisés comme critères pour rechercher des événements spécifiques. Pour notre contrat intelligent, nous n'utilisons pas d'arguments indexés, donc nous n'avons qu'un seul sujet. *gas* fait référence aux frais d'une transaction dans la blockchain Ethereum, les prix *gas* sont notés en `gwei`, qui lui-même est égal à 10^{-9} ETH. Le prix du *gas* n'est pas fixe, mais oscille entre 100 et 150 gwei au moment de la rédaction de ce manuscrit. Il est utilisé pour récompenser les mineurs qui incluent la transaction dans un bloc par le biais du processus de minage. Le prix du *gas* est utilisé pour réguler le nombre de transactions et éviter la congestion. Malheureusement, le prix d'une transaction sur Ethereum est extrêmement volatile et au cours de l'histoire d'Ethereum, les utilisateurs ont pu être témoins, à leurs dépens, d'augmentations démesurées de ces frais. Par exemple, en mai 2022, lors de la vente aux enchères des 55 000 parcelles NFT Otherdeed du projet de metaverse Otherside les prix du *gas* ont atteint jusqu'à 8 000 gwei pour se restabiliser quelques jours plus tard. C'est pourquoi, dans ce manuscrit, les estimations de coût que nous faisons ne sont valables que dans une courte fenêtre temporelle, néanmoins elles reflètent tout de même une tendance globale de coût sur Ethereum.

Concernant notre architecture, l'utilisation de notre smart-contract pour publier les informations sur une adresse IP qui a falsifié 1000 ID de nœuds différents, coûterait environ 6 000 000 d'unités de *gas*. À un prix du *gas* de 100 gwei, cela coûterait 0,667604 ETH qui, après conversion (au taux fixé au moment de la rédaction de cette thèse), revient à environ 2 700€. Il est important de noter qu'une transaction de base (simple transfert d'éther) coûte déjà 21 000 *gas*, qui sont convertis en 9€ (au même prix du *gas*). Nous considérons que ce coût est raisonnable au regard du service offert et par rapport à la capitalisation globale d'Ethereum (226 milliards de dollars au moment de la rédaction). Le service offert étant pour le bien commun, il pourrait être décidé in fine par la communauté d'appliquer un traitement spécial pour éviter tout frais.

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.11;

contract SybilGossip {
        address owner; //Address of the owner of the smart contract
        mapping(address => bool) allowedAddr; //Allowed to publish
        event Subnet(bytes[] subnet, bytes[][] last_bytes_port_nodeid);
        event Groups(bytes[] enodes);
        event Aliases(bytes[] enodes);

        constructor() {
                owner = msg.sender;
        }
        modifier onlyOwner() {
                require(msg.sender == owner, "Ownable: caller is not the owner");
                _;
        }
        //Add the given address to the list of allowed addresses
        function addUser(address _addressToAllow) public onlyOwner {
                allowedAddr[_addressToAllow] = true;
        }
        modifier isAllowed(address _address) {
                require(allowedAddr[_address], "You need to be allowed");
                _;
        }

        /*Publish the information of suspicious nodes in the same subnet
        _subnet: [subnet1, subnet2, subnet3, ...]
        _last_bytes_port_nodeid: [
        [byte_port_nodeid1_1, byte_port_nodeid1_2, ...],
        [byte_port_nodeid2_1, ...],
        [byte_port_nodeid3_1, byte_port_nodeid3_2, ...], ...
        ]
        */
        function SubnetPublish(
        bytes[] memory _subnet, bytes[][] memory _last_bytes_port_nodeid
        ) public isAllowed(msg.sender) {
                emit Subnet(_subnet, _last_bytes_port_nodeid);
        }

        /*Publish the information of nodes that are too close in the DHT
        _enodes: [enodes1, enodes2, enodes3, ...]
        */
        function GroupsPublish(bytes[] memory _enodes)
        public isAllowed(msg.sender) {
                emit Groups(_enodes);
        }

        /*Publish the information of nodes that have too many ID (aliases)
        _enodes: [enodes1, enodes2, enodes3, ...]
        */
        function AliasesPublish(bytes[] memory _enodes)
        public isAllowed(msg.sender) {
                emit Aliases(_enodes);
        }
}
```

Source Code 1 – Smart-Contrat permettant l'annonce de nœuds potentiellement Sybils

7.1.3 Discussion sur le remplacement du *smart contract* par un outil externe

Nous souhaitons motiver l'utilisation des journaux d'événements dans notre solution. Notre objectif est de proposer une application basée sur la blockchain qui soit pertinente en termes de coût, de performance et de facilité de développement. Wöhrer et al. [WZR21] ont exploré différentes conceptions d'architecture d'applications basées sur la blockchain.

Par exemple, dans le cas d'un système qui nécessiterait d'interagir avec des composants extérieurs à la blockchain, une approche hybride est intéressante à concevoir : l'application pourrait bénéficier de *l'event sourcing* qui fait référence au stockage des états comme une séquence d'événements immuables qui pourraient être rejoués à tout moment. Cette approche est très intéressante pour notre système : la liste des nœuds suspects trouvés par notre système de surveillance déclenche un événement provenant d'un *smart contract* qui est ajouté de manière immuable au répertoire d'événements ne nécessitant aucune confiance (*trustless event store*) de la blockchain Ethereum. En outre, Kostamis et al. [KSE21] ont étudié le coût et l'aspect pratique de différentes méthodes de stockage dans Ethereum. Le fait que les journaux d'événements soient stockés de manière immuable et qu'ils puissent être rejoués pour retrouver d'anciens états les rend intéressants à utiliser. Toutes les opérations exécutées par un *smart contract* ont un coût en *gas* (qui, pour rappel, est dérivé de l'Ether) mais l'utilisation des journaux d'événements est l'un des moyens les moins chers de stocker des données dans la blockchain Ethereum.

Ils ont décrit une autre façon de stocker des données, en utilisant la charge utile des transactions. C'est moins cher que les journaux d'événements, mais cela nécessite de stocker le hash des transactions dans une base de données externe pour retrouver un état ancien. Cela augmente la complexité de l'architecture et la base de données externe peut ne pas être aussi immuable que la blockchain Ethereum. Pour les raisons mentionnées ci-dessus, nous utilisons la blockchain Ethereum pour stocker et distribuer les informations sur les nœuds suspects.

Comme le montre la section 7.1.2, la solution que nous avons proposée pour annoncer des nœuds suspects à l'aide du système de contrats intelligents d'Ethereum est très coûteuse. On peut argumenter que nous pourrions utiliser un système basé sur InterPlanetary File System (IPFS) pour stocker la liste des nœuds suspects au lieu d'un coûteux *smart contract*. Ce système hybride est proposé par les travaux de [PDBU21] et utilisé en pratique par Rodrigues et al. [RSK+20] pour leur Blockchain Signaling System afin de prévenir les attaques par déni de service distribué. Mais, à notre connaissance, aucun fournisseur de stockage IPFS n'est gratuit (nous pouvons citer deux fournisseurs populaires non gratuits : FileCoin et Pinata Cloud). Une façon de contrôler les coûts serait de déployer notre propre cluster IPFS mais cela aurait quand même un coût non négligeable. Un autre inconvénient est que IPFS agirait comme un autre tiers de confiance dont la sécurité doit également être garantie. Nous pensons qu'il est plus sûr de ne pas introduire un point de défaillance externe dans le système. De plus, nous pensons qu'il ne serait pas éthique de s'appuyer sur un stockage externe alors qu'il existe une approche interne pour le stockage dans Ethereum et que sa capitalisation boursière s'élève à plus de 300 milliards de dollars. Puisque le service offert est pour la bonne santé du réseau mondial, il pourrait finalement être géré par la Fondation Ethereum comme une amélioration des protocoles.

7.2 Révocation des nœuds suspects

7.2.1 API REST et appel RPC pour la révocation

Les nœuds suspects découverts lors d'un crawl sont exposés via une API web dont le code source est disponible dans le dépôt Crawleth [Eisa]. Cette API est disponible à cette adresse :

`http://crawleth.loria.fr:5000/api`. Nous avons développé deux programmes open-source rassemblés au sein d'un projet appelé « Sybil Prevention » [32] [Eisc]. Le premier programme, « Sybil Gossip » est utilisé pour faire le pont entre l'API mentionné plus tôt et la blockchain en envoyant les informations de ces nœuds à notre *smart contract*. Ainsi, après chaque crawl, jusqu'à trois événements (« Subnet », « Groups » et « Aliases ») sont déclenchés avec les informations de ces nœuds. Le site web exposant l'API propose également des statistiques et des graphiques en direct sur le réseau P2P à partir de sa supervision continue, ainsi que l'historique des explorations précédentes (agrégées par heure pour le jour en cours, et par jour sinon).

Le deuxième programme, "Sybil Revocation", a été développé pour récupérer les journaux d'événements, vérifier l'existence des nœuds suspects et supprimer toute connexion active d'un client Ethereum local après vérification. Il est représenté dans la partie droite de la figure 7.1. Tout utilisateur qui a déployé un nœud Ethereum peut utiliser ce programme pour empêcher son nœud de maintenir une connexion à un nœud suspect. Cette application utilise l'API JSON-RPC (protocole d'appel de procédure à distance codé en JSON) du client officiel de Geth. Plus précisément, il utilise l'appel "Admin.removePeer" qui est non documenté mais qui existe depuis 2016 [Thel]. Il peut supprimer toute connexion active (entrante ou sortante) à un pair spécifique. Comme il n'existe pas encore d'appel RPC pour imposer la mise sur liste noire d'un nœud, notre outil doit vérifier périodiquement si les clients Ethereum locaux qu'il protège ont des connexions à des nœuds suspects connus. Ce programme est une preuve de concept, une implémentation directement dans un client Ethereum rendrait le processus de révocation plus efficace en nettoyant également la table de routage et en empêchant les contacts suspects d'être partagés entre pairs. Cette implémentation dans les clients Ethereum permettrait également de réduire les coûts intrinsèques liés à l'utilisation du *smart contract* qui stocke une quantité de données non négligeable via les journaux d'événements en permettant peut-être une utilisation privilégiée à coût réduit de ces fonctions.

7.2.2 Performances de Sybil-Prevention

Pour démontrer la viabilité de notre architecture, nous avons réalisé l'expérience finale suivante sur un réseau de test Ethereum. Nous avons connecté un pair non modifié dans le réseau et exécuté notre outil de révocation RPC en parallèle. Ensuite, nous avons établi manuellement des connexions vers 10 pairs connus pour être détectés comme suspects dans les crawls précédents parce qu'ils portent trop d'alias (ce qui constitue la principale menace actuelle pour le réseau que nous avons mis en évidence dans le chapitre 6). Nous avons effectué un nouveau crawl et exécuté le contrat intelligent susmentionné pour ajouter les pairs suspects que nous avons trouvés dans les journaux (comme cela a été fait sur un réseau de test Ethereum, aucun frais n'a été facturé). Enfin, l'outil de révocation RPC a récupéré avec succès la liste, effectué la vérification et coupé la connexion aux 10 pairs, validant ainsi l'ensemble du processus.

Pour évaluer les performances de notre système, nous avons également mesuré le temps nécessaire à chaque étape de notre architecture, de la surveillance à la révocation. Les résultats sont donnés dans le Tableau 7.1. Un nouveau crawl est effectué toutes les 15 minutes environ (les résultats sont agrégés toutes les heures sur `http://crawleth.loria.fr:5000/api/latest/deviant/all`), puis les résultats sur les nœuds suspects sont publiés dans la blockchain à l'aide du *smart contract*. Cette étape ne prend qu'une seconde sur le réseau de test mais peut prendre jusqu'à plusieurs minutes sur le réseau Ethereum principal à cause du temps de minage des transactions [33]. La liste des nœuds suspects est alors instantanément extraite des journaux d'événe-

32. `https://gitlab.inria.fr/jeisenba/sybil-prevention`

TABLE 7.1 – Temps d'exécution de chaque étape de notre solution

Opération	Temps d'exécution
Nouveau crawl	∼15 minutes
Annonce des nœuds suspects	dépend du temps moyen de minage des transactions : - dans notre expérience sur le réseau de test < 1 s - sur le réseau principal < 5 min
Récupération des nœuds suspects via les journaux d'événements	< 1 s
Vérification des nœuds prétendument suspects	< 5 s
Révocation	< 1 s

ments et le programme de vérification de "Sybil Revocation" démarre. Il vérifie que les nœuds publiés existent réellement et sont effectivement suspects en émettant des requêtes FINDNODES sur les alias trouvés pour un nœud jusqu'à ce que le seuil de détection soit vérifié. Ce processus de vérification dure environ 3 à 4 secondes par alias. En parallélisant le processus de vérification, il dure le même temps pour tous les alias qui doivent être vérifiés.

Une fois que les nœuds ont été vérifiés, le processus de révocation est instantané. À partir du moment où l'information des nouveaux nœuds suspects détectés parvient aux pairs, l'ensemble du processus est réalisé en quelques secondes. Le principal facteur limitant est le temps nécessaire pour effectuer une exploration, mais comme il faut du temps pour insérer des nœuds malveillants dans le réseau, la véritable clé est d'effectuer les explorations en continu pour assainir le réseau. Le temps d'exploration peut également être réduit en divisant l'espace d'adressage à explorer entre plusieurs crawlers de confiance, cette étape étant facilement parallélisable.

Conclusion

Nous avons proposé dans ce chapitre une solution qui permet de prévenir des attaques Sybil contre la DHT d'Ethereum. Pour cela, nous avons développé un projet appelé "Sybil-Prevention" qui permet, à l'aide de la publication des nœuds suspects via l'API web de Crawleth et de leur diffusion via la blockchain, de révoquer des nœuds de manière complètement distribuée après des vérifications locales. Ces vérifications sont essentielles et permettent de réduire fortement la confiance accordée au crawler et ainsi prévenir les conséquences d'un détournement potentiel qui pourrait nuire au réseau en faisant accuser et révoquer à tord des pairs légitimes. En revanche, une collusion crawler-attaquant, bien qu'improbable car l'écriture dans le smart-contract est authentifiée, permettrait de mener une attaque sybil silencieuse et rendrait caduque les mécanismes subséquents, ce qui reviendrait à la situation actuelle.

Notre architecture et programmes s'avèrent performants pour révoquer les connexions à des nœuds suspects. En effet, l'utilisation d'un smart-contract couplé à un stockage des annonces

33. `https://ethgasstation.info/`

dans les journaux d'événements permettent de réduire le temps d'exécution de la publication et récupération des informations en quelques secondes seulement. Le système dans sa globalité est restreint par le temps entre deux explorations du réseau par Crawleth. Ces 15 à 20 minutes en moyenne pour un crawler peuvent être réduites en répartissant le crawl de l'espace d'adressage entre plusieurs instances.

Cependant, notre solution a un coût non négligeable en Ether, mais n'utilise aucun service de stockage d'informations tierce-partie garantissant l'indépendance de la protection du réseau P2P. Ce coût pourrait être réduit en considérant que l'apport de notre solution est pour le bien commun et donc pourrait bénéficier d'une exemption de frais ainsi qu'une implémentation directe dans les clients Ethereum.

Conclusion Générale

Les blockchains, notamment celles basées sur des réseaux P2P publics telles que Bitcoin et Ethereum, sont des systèmes très critiques aujourd'hui à cause de leur rôle croissant dans l'économie (des centaines de milliards d'euros de capitalisation) et l'usage des smart contracts (NFT, finance décentralisée, organisation autonome décentralisée, etc). Les réseaux P2P sous-jacents ont un rôle essentiel, mais peuvent souffrir de plusieurs problèmes qui réduisent leur fiabilité et robustesse. Un trop faible nombre de pairs, une répartition géographique déséquilibrée, une centralisation du point de vue du réseau IP, un fort taux d'attrition sont autant de problèmes pouvant les affecter. Ils peuvent également être victimes d'attaques complexes telles que l'attaque Sybil ou Eclipse qui peuvent tirer parti des problèmes cités précédemment. Ces réseaux P2P peuvent également être utilisés pour optimiser les échanges et le stockage des données au sein des blockchains, notamment lorsqu'ils sont basés sur une DHT, comme cela est le cas pour Ethereum.

Dans le cadre de cette thèse, nous avons réalisé trois principales contributions. Nous avons implanté un crawler pour la **supervision des réseaux P2P de Bitcoin et Ethereum** et proposé une étude des caractéristiques de ces réseaux. Nous avons ensuite proposé une **nouvelle architecture de stockage des données d'Ethereum** basée sur la répartition du stockage de certaines données à l'aide de sa DHT, réduisant considérablement les besoins en termes d'espace de stockage des pairs, et donc du système dans sa globalité. Pour finir, nous avons également mis en exergue des comportements suspects de type Sybil sur le réseau P2P Ethereum, pouvant potentiellement mettre en défaut, entre autres, notre nouvelle gestion du stockage. Nous avons donc proposé une **architecture de prévention des attaques Sybils** composée d'un système de supervision et d'un système de révocation complètement distribué.

1 Travail réalisé

Dans un premier temps, nous avons montré dans l'état de l'art que les réseaux P2P de Bitcoin et Ethereum sont finalement très similaires : sélection des voisins, topologie P2P non structurée (malgré la DHT d'Ethereum), propagation des messages par inondation, nombre de pairs. Les caractéristiques de ces deux réseaux P2P, telles que le taux d'attrition, la centralisation du point de vue du réseau IP, la distribution géographique ou la popularité des pairs dans les tables de routage restaient cependant assez peu documentés de manière rigoureuse, notamment lorsque l'on considère le réseau dans son ensemble.

Pour analyser ces caractéristiques, nous avons utilisé respectivement le crawler Bitnodes (que nous avons modifié) et conçu et développé notre propre crawler, Crawleth, pour mener des campagnes de mesures de plusieurs mois sur les réseaux P2P de Bitcoin et d'Ethereum. Les deux réseaux P2P sont comparables en matière de taille, de taux d'attrition, de distribution géographique ou encore stabilité des pairs, mais ils se démarquent sur un point important, à

savoir la centralisation au niveau du réseau IP des pairs. En effet, 70% des adresses IP des pairs d'Ethereum sont concentrées dans seulement 10 ASes (*Autonomous Systems*), dont 9 d'entre eux sont des fournisseurs de cloud, alors que pour Bitcoin cette proportion descend à 35%.

Ensuite, nous nous sommes intéressés au stockage des données de la blockchain Ethereum. Le réseau P2P de celle-ci est construit autour d'une DHT dont la capacité à indexer et stocker des données de manière distribuée est largement inexploitée. Par leur fonctionnement, les clients actuels classent les données en deux catégories et dans des bases différentes, à savoir les données les plus récentes (90 000 derniers blocs) et celles plus anciennes. Ces dernières ne peuvent plus être modifiées et n'ont plus vocation à être accédées fréquemment depuis l'introduction d'une nouvelle structure de données visant à accélérer la synchronisation d'un nouveau nœud. Nous avons donc proposé une nouvelle solution de stockage permettant de gagner environ 95% de stockage sur les données anciennes (ce qui représente environ 58% de gain sur le stockage total d'un client). Cette nouvelle solution tire parti de la DHT et de la répartition de la charge de stockage de manière homogène sur l'ensemble des pairs qui composent le réseau P2P. Elle est en outre déployable incrémentalement et pleinement compatible avec le fonctionnement actuel des clients.

Pour finir, notre état de l'art a montré qu'une DHT est par défaut vulnérable à des attaques Sybil permettant à un attaquant de prendre le contrôle des données indexées en son sein. Nous avons donc étendu la supervision du réseau P2P à la détection de comportements suspects de type Sybil et avons confirmé la vulnérabilité du réseau P2P d'Ethereum à ce type d'attaque, constatant que seulement quelques adresses IP pouvaient détenir plus de 10 000 identités. Nous avons conçu et développé un mécanisme de révocation locale complètement distribué basé sur une supervision en temps réel du réseau et la détection des pairs suspects par un ou plusieurs crawlers. Nous avons couplé cette supervision à un smart contract permettant d'enregistrer et de diffuser leurs informations et permettre à chaque client du réseau de procéder à leur révocation après vérification.

La liste complète de nos productions est décrite dans les sections Publications et Implantations et jeux de données collectés. Pour chacune de ces contributions, l'ensemble de nos programmes et nos jeux de données sont disponibles publiquement afin de rendre nos résultats entièrement reproductibles.

2 Perspectives de recherche

Les résultats présentés dans cette thèse ouvrent des perspectives de recherche à court et long terme.

Amélioration de la supervision de Bitcoin et Ethereum

Tout d'abord, une piste de travaux futurs concerne la supervision des nœuds Bitcoin. En 2019 et début 2020, les nœuds IPv4 représentaient environ 75% du réseau alors que les nœuds IPv6 et Tor représentaient respectivement 10% et 15% du réseau. En observant en 2022 les statistiques sur le site `https://bitnodes.io`, on remarque que la part des nœuds IPv4 a considérablement diminué, au profit de nœuds Tor. Par exemple, en mai 2022, 38 % des nœuds fonctionnaient en IPv4, 55 % des nœuds utilisaient Tor, et 7 % utilisaient IPv6. À l'époque où nous menions ces travaux sur Bitcoin, il était raisonnable de considérer que la tendance globale du réseau IPv4 pouvait être extrapolée à l'ensemble du réseau, mais cela n'est dorénavant plus le cas. En intégrant les nœuds utilisant Tor, la caractérisation des pairs actuels du réseau serait plus représentative et les propriétés du réseau global qui en découlent plus précises.

Pour Ethereum, notre outil de supervision ne permet pas de caractériser précisément le client utilisé par les pairs du réseau P2P. Pour cela, les couches protocolaires au-dessus du protocole de découverte des nœuds devraient être implantées dans Crawleth, notre outil de supervision. Cette caractérisation des clients permettrait de bien différencier les clients Ethereum des autres et de mettre en exergue d'autres vecteurs d'attaques potentiels du réseau P2P, par exemple la faiblesse d'une partie du réseau à des vulnérabilités logicielles connues de certains clients. Cela permettrait, en outre, de conduire une analyse détaillée des comportements des pairs, en particulier des nœuds identifiés comme suspects, en échangeant des messages applicatifs avec eux. Des comportements de censure (refus de propager des transactions/blocs dans une partie du réseau) pourraient par exemple être ainsi découverts.

Une autre possibilité pourrait concerner la détection de super-pairs, à savoir des nœuds spéciaux très connectés (c'est-à-dire avec un très fort degré de connectivité) dans le réseau. En effet, ces super-pairs pourraient servir à avantager certains mineurs dans le cadre d'une blockchain basée sur la preuve de travail comme Bitcoin. À défaut de pouvoir inférer la topologie précise du réseau P2P, il devrait être possible d'inférer l'existence de super-nœuds à partir de données de sondes placées dans le réseau et étudier la propagation des blocs et transactions. Une des difficultés d'une telle méthode est de s'assurer que les sondes sont placés stratégiquement dans le réseau pour pouvoir inférer les connexions à travers les délais mesurés.

Extension à d'autres systèmes de blockchains

Les réseaux P2P d'autres systèmes de blockchains prometteuses, comme Tezos [34] ou Zcash [35], ont eu très peu d'attention de la part de la communauté scientifique. La force de Tezos réside dans son mécanisme de consensus basé sur une preuve d'enjeu native et son langage de smart contract Turing-complet, purement fonctionnel et adapté à la vérification formelle. Zcash permet quant à elle une protection de la vie privée accrue par son nouveau type de transaction basé sur des preuves à divulgation nulle de connaissance. La supervision et l'étude des réseaux P2P sous-jacents de ces deux blockchains seraient intéressantes pour la communauté.

Système de supervision entièrement distribué

La principale faiblesse de l'architecture de prévention des attaques Sybil décrite dans le chapitre 7 réside dans le caractère peu distribué de la tâche de supervision. Bien qu'il soit possible de déployer plusieurs instances d'un crawler et de les coordonner, la logique pair-à-pair qui prévaut aux blockchains demanderait une distribution complète de cette fonctionnalité. Une piste d'amélioration consisterait donc à distribuer entièrement la tâche de supervision sur l'ensemble des pairs du réseau. Dans un réseau d'overlay utilisant une DHT, chaque pair peut devenir responsable de la supervision de son voisinage immédiat, dont il a une très bonne connaissance, et donc à moindre coût, puis utiliser un smart-contract pour diffuser les informations. Cependant, la supervision pouvant être perturbée par des nœuds malveillants, il faudrait impérativement y adjoindre un mécanisme de consensus spécifique pour la rendre robuste, ce qui reste à concevoir aujourd'hui. Un autre défi est de limiter les échanges d'informations de supervision entre pairs afin de ne pas surcharger le réseau.

34. `https://tezos.com/`
35. `https://z.cash/`

Complémentarité du stockage dans la DHT et *Sharding*

Les développeurs d'Ethereum ont annoncé l'arrivée (courant 2023) d'une nouvelle fonctionnalité pour réduire la capacité de stockage nécessaire pour un client Ethereum, appelée *Sharding*[36]. Ce concept est emprunté aux architectures de bases de données et son implantation à venir dans Ethereum est peu documentée. Il semblerait que cela consiste à découper la blockchain en *shards* (morceaux) et les répartir au sein du réseau P2P. La complémentarité de cette nouvelle fonctionnalité et de notre proposition de stockage dans la DHT serait une piste intéressante à étudier.

Efficacité énergétique du réseau P2P

Ethereum a changé son mécanisme de consensus, basé auparavant sur la preuve de travail, par un mécanisme basé sur la preuve d'enjeu depuis septembre 2022. Ce changement a été motivé, entre autres, par les gains en matière de dépense énergétique. En effet, la preuve de travail est un très fort facteur de dépense énergétique[37]. Les développeurs d'Ethereum ont annoncé qu'un changement de PoW vers PoS apporterait un gain de ∼99.95% par rapport à sa consommation énergétique[38]. Un autre levier permettant de baisser davantage les dépenses énergétiques d'Ethereum est l'optimisation des échanges au niveau du réseau P2P. L'idée est de réduire les dépenses liées aux propagations des messages vers des pairs très éloignés géographiquement. En effet, la sélection des voisins pour un nœud Ethereum est actuellement purement aléatoire. Un nœud européen peut ainsi être connecté à un nœud asiatique ou américain. Une piste de réflexion pourrait être de modifier la règle de sélection pour introduire la notion de latence, de proximité réseau (au sein d'un même AS) et de distribution géographique pour trouver un compromis entre la résilience du réseau et l'optimisation de la propagation des messages. Ce concept rejoint le problème de la localité dans les systèmes distribués.

36. `https://ethereum.org/en/upgrades/sharding/`
37. `https://ccaf.io/cbeci/index`
38. `https://blog.ethereum.org/2021/05/18/country-power-no-more`

Productions

Publications

Les différentes contributions présentées dans cette thèse ont donné lieu à diverses publications scientifiques :

- **An Open Measurement Dataset on the Bitcoin P2P Network**, publié à IFIP/IEEE IM 2021 [ECP21b], présente le jeu de données sur le réseau P2P de Bitcoin que nous avons collecté, ainsi que quelques résultats préliminaires. Ces éléments sont présentés dans le chapitre 3
- **A Comprehensive Study of the Bitcoin P2P Network**, publié à BRAINS 2021 [ECP21a], présente une étude détaillée des caractéristiques du réseau P2P de Bitcoin à partir du jeu de données de la publication précédente. Ces résultats sont présentés dans le chapitre 3
- **Ethereum's Peer-to-Peer Network Monitoring and Sybil Attack Prevention**, publié dans le journal de Springer JNSM Octobre 2022 [ECP22], présente notre outil open-source de supervision du réseau P2P Ethereum ainsi que l'étude de diverses métriques pour caractériser le réseau P2P (chapitre 4). Cette publication présente également une étude de métriques permettant de déceler les pairs suspects (chapitre 6), faisant penser à une attaque Sybil. Elle présente enfin notre architecture de supervision et de révocation de ces pairs suspects (chapitre 7). Tous les outils sont open-source et les jeux de données publiquement accessibles.

Implantations et jeux de données collectés

Toutes les implantations sont open-source et disponible publiquement sur diverses plate-formes :

- **Bitnodes**, notre fork et modifications sont disponibles sur Github [Eisb]. Il s'agit de l'explorateur du réseau P2P Bitcoin écrit en Python.
- **Geth modifié**, notre preuve de concept du stockage efficace des données à long terme de la blockchain Ethereum est disponible sur Github [Eis22]
- **Crawleth**, notre explorateur original développé en Python du réseau P2P d'Ethereum est disponible sur l'instance Gitlab d'Inria [Eisa].
- **Le site web de présentation des données de supervision de Crawleth**, également développé en Python est consultable à cette adresse `http://crawleth.loria.fr:5000/`. L'API web pour récupérer les données des comportements suspects est également disponible sur ce site web.
- **Sybil-Prevention**, notre outil en Python pour vérifier les comportements suspects sur le réseau P2P Ethereum et permettre éventuellement la révocation des pairs fautifs est disponible sur Gitlab [Eisc].

Nos deux jeux de données sont également disponibles publiquement : `https://concordia-btc-p2p.lhs.loria.fr/index.html` et `https://concordia-eth-p2p.lhs.loria.fr/index.html`

Table des figures

Liste des tableaux

Liste des algorithmes et programmes

Annexes

Annexe A

Extrait du jeu de données généré par notre instance de Bitnodes

L'ensemble du dataset : `https://concordia-btc-p2p.lhs.loria.fr/index.html`
Dossier crawl, fichier *date.json* :


```
[
  [
    ["19f05194d708031537c86be207d746661dff660afdc3b7df4792b4d941a6c7e2","85.25.134.0"],
    8333,
    1037,
    617492
  ],
  [
    ["d472c7ed07982e5079904f116af4691d9f61a653d51c49957a73789a4846bf5a","206.189.138.0"],
    8333,
    1036,
    617492
  ],
  [
    ["fff3c5285b9b3eba77d64d1a509cb786bb1395db6c2fd46fc08b9b9e1323fbe1","68.36.95.0"],
    8333,
    1037,
    617492
  ],
  [
    ["0b8a47c3da6a9fd7a51c31d06e52bc305eece56e9228c992bc8cbb21256816ac","176.191.182.0"],
    8333,
    1033,
    617492
  ],
  [
    ["c1a613c8594d6047dfbe264bf0c431ffef3fb7efda94efff646b43f4d9a78a19","202.186.228.0"],
    8333,
    1033,
    617492
  ],
  ...
]
```

Dossier crawl, fichier *nodes_per_getADDR_date.csv* :

```
1, f4d8a95c858d1a1cda33673bbf561d606116a47a487fd361934db55f81a7e2ce-35.185.131.0-8333-1037,
↪  289, 207, -1, 262, 289, 279
2, 5ed7d2dd1e26459f5e55e7bc355de44935d553e08058e3c7e1385926abf6e762-104.199.192.0-8333-1037,
↪  274, 136, -1, 274
3, cddacda71254da638a21f0de804687342c3161b7c0de2ac7d568d95a7e882496-54.245.167.0-8333-1037,
↪  -1, -1, -1, -1
4, 2d949269793bbc380182f6cb260ecc073c30ab83eba9c6dfe7cf5f62ce02eeb1-3.24.124.0-8333-1033, -1,
↪  -1, -1, -1
5, 9f9f5a8552dab50f87d817888fbe04c26fc386c698a474864d57439e88a36ef8-157.230.144.0-8333-1037,
↪  -1, -1, -1, -1
6, b57dc07072d96e0a6458ca58cf782be4d63f22231fa9f388bf27a461a45f913b-136.49.127.0-8333-1033,
↪  -1, -1, -1, -1, -1, -1
7, c8a816449720750b0a76634c30fce89c6e3bf5002f5389b600947c5a767ce48c-169.48.179.0-8333-13, -1,
↪  -1, -1
8, 57f1c5bc36663b12dd7872c2e76784b303bff489494444dc52a80921d1caf09f-178.209.113.0-8333-1, 75,
↪  73, 75, 74, 69
9, 8bf7f08e5c0ce2326170b6bf06d9bbc78310f65baa96b99c3577bec26aec7133-195.154.253.0-8333-13, -1,
↪  -1, -1, -1
10, a0700331a460858ba3426140a5c004758d1656321596f5a39c51e7785f7099a3-185.162.128.0-8333-1037,
↪  -1, -1, -1
11, 924f4f2c66fb95b4d7b29d079ff7343ed9d19ce75d010bc758c0d756606ac494-165.22.154.0-8333-1037,
↪  -1, -1, -1
12, 2c2c449057a06355cc88e96256136e6ea2005428f7f66d16139da72522c7f96b-47.89.15.0-8333-1037, -1,
↪  -1, -1
13, 7fa58e7e712fcf3a093285dec8b1d191c91bb92636bda9a606b4a2240ff8acce-85.214.212.0-8333-1037,
↪  -1, -1, -1, -1
14, 634fcd048e1ec07086f0275ba79326aaff622f2593266b0aae5c0e994ce5b14c-52.221.241.0-8333-13, -1,
↪  -1, -1, -1
15, 20a14aaf46c0d188ea84bb3bcf8d5c300f5f21a4834aeb175bfe5a0df161f92c-93.43.96.0-8333-1037,
↪  111, 111, 111
16, c6a0e73f9d124920a96e263b5938544c3b8958ec1238f5497b609bb927bf612a-142.93.137.0-8333-1033,
↪  163, 100, 138, -1, 163
17, d6742dae59f6c5247118e9ec8ae080a70463d70b40bf9862bd2cb7a2b260ff65-144.202.110.0-8333-1036,
↪  127, 80, 113, -1, 127
...
6823, 18ab240978da964c6c43ebf87a7a5132d4893672798e4f2e32feec80d485bd59-45.79.190.0-8333-1036,
↪  133, 87, 133, 130, -1
6824,
↪  468e9c64ab11f2d6cae368cdc3e252cb88a2b77564f01e35be7af4bbc34d0568-178.128.202.0-8333-1037,
↪  221, 221, 221
6825, 3cb1749a419edaa311a7ab5f27ab66d1d471ded043d4b30443a999db48da18ec-74.83.193.0-8333-1037,
↪  -1, -1, -1
6826,
↪  8b6921ab6ae7c134c84a422f7aad8d2dac58493005c8405e83cb304644e126e2-158.177.203.0-8333-1037,
↪  -1, -1, -1
6827, 6daaa31acbb637f961430507204aaeb654f118bb6f01a1391d6d22850ea74349-45.116.165.0-8333-1037,
↪  -1, -1, -1
6828, 4382360c91297756cbe465eca5fea71818643e31d437283da79bc876ce43e0bf-34.84.109.0-8333-1037,
↪  240, 240, 240
6829, e57a98fedebcaa03b04feea1ce8e1e0be73c5c4032a61a8456d8d6dd7755a2c7-91.4.53.0-8333-1033,
↪  -1, -1, -1
6830, b7e3d4db60e61b4460794b73c13241465395bf4287c33e6e8116f33805157b66-212.51.139.0-8333-1033,
↪  97, 97, 97
...
```

Dossier crawl, fichier *up_nodes_per_seconds_date.csv* :

```
0,8
5,612
10,393
15,349
20,1226
25,526
30,272
35,313
40,525
45,218
50,247
55,338
60,217
65,191
70,290
75,167
80,146
85,195
90,165
95,125
100,176
105,127
110,101
115,122
120,104
125,64
130,61
135,49
140,0
145,2
150,4
155,1
160,0
165,0
```

Dossier export *date.json* :


```
[
  [
    ["083e2504d6928e1b56761fd56b532635066019f6d31fa7aee9c0a7147f460162","34.243.101.0"],
    8333,
    70015,
    "/Satoshi:0.18.1/",
    1581005548,
    1037,
    617492,
    "ec2-34-243-101-178.eu-west-1.compute.amazonaws.com",
    "Dublin",
    "IE",
    53.3338,
    -6.2488,
    "Europe/Dublin",
    "AS16509",
    "Amazon.com, Inc."
  ],
  [
    ["3da5610102c0808fa564870a92562820163d8d2bfaf7e1a2c151823cbaffb182","62.213.214.0"],
    8333,
    70015,
    "/Satoshi:0.18.0/",
    1581006197,
    1037,
    617492,
    "62-213-214-207.ip.stuart.be",
    null,
    "BE",
    50.85,
    4.35,
    "Europe/Brussels",
    "AS28707",
    "Kangaroot BVBA"
  ],
  [
    ["d5923a0a3c26415796673d5915bf39a77abb634324c2aef30e5ec11b2fada862","116.202.116.0"],
    8333,
    70015,
    "/Satoshi:0.18.1/",
    1581005548,
    1037,
    617492,
    "static.247.116.202.116.clients.your-server.de",
    null,
    "DE",
    51.2993,
    9.491,
    "Europe/Berlin",
    "AS24940",
    "Hetzner Online GmbH"
  ],
  [
    ["4f6f326a1ef01a6ece528c0f215ef22a64cb35037330141085eb03ec1f7f53a9","99.85.80.0"],
    8333,
    70015,
    "/Satoshi:0.18.1/",
```

```
    1581421600,
    1037,
    617492,
    ["4f6f326a1ef01a6ece528c0f215ef22a64cb35037330141085eb03ec1f7f53a9","99.85.80.0"],
    "Melbourne",
    "US",
    28.2062,
    -80.6874,
    "America/New_York",
    "AS7018",
    "AT&T Services, Inc."
  ],
  [
    ["e5f890e0ab4850ac55bf3b0642893d370413591de9fc86a6b1fa1c1c1a8af0bf","176.121.14.0"],
    8333,
    70015,
    "/Satoshi:0.18.0/",
    1581005548,
    1036,
    617492,
    ["e5f890e0ab4850ac55bf3b0642893d370413591de9fc86a6b1fa1c1c1a8af0bf","176.121.14.0"],
    null,
    "UA",
    50.45,
    30.5233,
    "Europe/Kiev",
    "AS210138",
    "Flowspec Ltd"
  ],
  [
    ["89f314590240430160e50fd2bf34b0370aac6b94cec3ccf3d9e7eef2949de8c3","104.248.157.0"],
    8333,
    70015,
    "/Satoshi:0.17.1/",
    1581005548,
    1036,
    617492,
    ["89f314590240430160e50fd2bf34b0370aac6b94cec3ccf3d9e7eef2949de8c3","104.248.157.0"],
    null,
    "SG",
    1.314,
    103.6839,
    "Asia/Singapore",
    "AS14061",
    "DigitalOcean, LLC"
  ],
  ...
]
```

Annexe B

Extrait du jeu de données généré par notre instance de Crawleth

L'ensemble du dataset : `https://concordia-eth-p2p.lhs.loria.fr/index.html`
Fichier *raw_ aggregated_ results_ date.json* :


```
{
  "stats": {
    "aggregated_crawls": 5,
    "original_filenames": [
    "2021-09-01 22:14:56.014.json",
    "2021-09-01 22:26:13.660.json",
    "2021-09-01 22:38:17.239.json",
    "2021-09-01 22:49:49.859.json",
    "2021-09-01 23:01:43.355.json"
    ],
    "up_count": 16311
  },
  "up": [
  {
    "IP address": [
    "d9ed4fa8f52da19bfdb3c70efe79a30f7d3c379b3b261edec2a72f8c90f61205",
    "35.170.120.0"
    ],
    "UDP port": 30303,
    "Seen node IDs": [
    "0xb9a5e28f048ea6b1a81063f1527b1cb225a8d8d3c241e7a16d030d355f8382c600f081c1ff52b88f6cecbd
    ↪  5bae7bb0951ae41b0bb6aab14ca9be6d97ea642234"
    ],
    "as": "AMAZON-AES",
    "asn": "AS14618",
    "city": "Ashburn",
    "country": "United States",
    "country_code": "US",
    "latitude": 39.0469,
    "longitude": -77.4903
  },
  {
    "IP address": [
    "eb09ca88ca4053eda15879cb18264c52ec7729679da02d8e9fe4b1fc804cb99c",
    "95.217.211.0"
    ],
```

```
    "UDP port": 30305,
    "Seen node IDs": [
    "0x35cdd06f14e8dad12ebeff4dccaae2293f0688ac727a66743266be08ca157bbeb70c95a83cb6f5cb40b9c7
    ↪  bc17cb10cc61059691061d116f51ac6984753a0757"
    ],
    "as": "Hetzner Online GmbH",
    "asn": "AS24940",
    "city": "Helsinki",
    "country": "Finland",
    "country_code": "FI",
    "latitude": 60.1719,
    "longitude": 24.9347
  },
  {
    "IP address": [
    "410e581409a3be9dc66aba3b003429aa858a72adb816352168fc2c21bee3971e",
    "172.104.24.0"
    ],
    "UDP port": 30303,
    "Seen node IDs": [
    "0x89e046a4f10c64265941789b2e3be900adf5132ced13756aeea126cf59b516445ed8053b600aa764860f1a
    ↪  ad552f4f4f3b4250c59b3b8a84ead3d3527c005606"
    ],
    "as": "Linode, LLC",
    "asn": "AS63949",
    "city": "Cedar Knolls",
    "country": "United States",
    "country_code": "US",
    "latitude": 40.8229,
    "longitude": -74.4592
  },
  {
    "IP address": [
    "122a56272f2280b9a58ab83f9d5587243402fe47af586703ee8480847192da23",
    "35.75.124.0"
    ],
    "UDP port": 30303,
    "Seen node IDs": [
    "0x362b7d8a3e148498f28d13b43d38a5db9435b59a79015c5aa4024344132ef56bf9240581ee328420fc32c9
    ↪  13c4a908e4cb733c0627a40dc73117aeb0d7a3428f"
    ],
    "as": "AMAZON-02",
    "asn": "AS16509",
    "city": "Tokyo",
    "country": "Japan",
    "country_code": "JP",
    "latitude": 35.6887,
    "longitude": 139.745
  },
  ...
}
```

Annexe C

Extrait du jeu de données intermédiaire pour la detection des nœuds suspects sur Ethereum

Fichier *aliases_ date.json* contenant les nœuds possédant un trop grand nombre de NODEID (ici seuil fixé à 2) :


```
{
  "threshold": 2,
  "results": {
    "IP_address": [
      "0xcc947b8d12482875c81615616ffdff1ae64251c2b3b6430de9c56c5a7473d1ca63d0c4e463780d81ba94
      ↪  19abedf2c2fcec8b2a0957381fb1be619bfc05ede65a",
      "0x9d503a1401ad1fac18422964d7c5bdcf978b5b237772a2abd393590696faa35cbd403a9f89670bc3abe3
      ↪  b2e61eb6cb75fa8bcf44ea4b83d0aa33ad59c692842d",
      "0x2078287860662475a4c2b38a72a98c6068969eacfdbfb32e66b60d846efa62d2a8b71425981305ac1976
      ↪  b0ecf2696e64aad5688e55611b478a79c8b81899e619",
      "0x25841d507fc5d04be63b6c4f664c70d5c94237dc487e1cd56fbb4a8616d93a2e3a8d8c116eafe16ad29b
      ↪  b6ff5b32efdf0d4c4203b2942f18b348f26ed4638760",
      "0x98569a6b090836e4d74a2d653698a9d3ad03942e5751cd744b5b329203ea0beb6d5760a9182c1b63a640
      ↪  459eefcc95b270588fe1d359dda7c17b52ddf5bcbf57",
      "0xe92036ee636998be7a8fa5111a696e73cac62a5bf99e9e6a0ce68eb20e450248cfb26851b855d1765d4a
      ↪  beaf8106c4c1c7b38b8bdecceefdf4b3196000dabda9",
      ...
    ],
    "IP_address": [
      "0x0afc68a416acdcc56f56d926f001703fb8d4610c5e07ee9cd111401332e77963a4360fc841e37b3abd7a
      ↪  64ee4d35a8bbc311920c4ce1fca44f0545b0e0bd5727",
      "0xadfc8d77eb7dbf18be59bb7c369d209eb83c9b9903cced6f0d973cba768913530f713b56ecea82305864
      ↪  c89f040f4a76b5c102a143b5c9565881181875f769b8",
      "0x5782bc96b5097668ac5b032bf4130b1dec36d9bea2ff02a03ee1a8af5f35ef9b9176b37de94dab821f3b
      ↪  45a8c65b4c4ff445f20caa72b67aac8b7507f37972ba",
      "0x7c65154329d87c83036d5e2b66ada4b1bd3e9613a9418336719605cc76c0305a0ed41e65501a2f72135b
      ↪  230164f6d8a14da2cfc935d6d104ad74e9c6cb3a21b8",
      "0xfd41d98ce3da85c5e0a06516e05c828f2ad8c83bf5d8b7b78954e1e94819e24a7b2b8fc3687dc12e6ee3
      ↪  fca68f3d1a61493125d014c0144837a8bb5a98450fbd",
      "0xe3aaef0d5f9619e4943a175d727eff0ed75b9db8fb97bf8a81e3a85c6ea621f0e4bcd2a7df30e3529d19
      ↪  af413d785ea4533bf2a81495332b037751798c887cc8",
      ...
    ],
    "IP_address": [
```

```
      "0x48e59d370fb01f8f230fdfe6d65554eef16324824cea34228f58c1134d19e25d1d76503b8250facd2958
      ↪  8ce5884601c98c68f52b07da6a368ba7885c6feac889",
      "0x1dedf8b22f2cb5d406b64f4204d8b351473f04c35279fdd938a6368531154f8c84ddabb83158cafcae37
      ↪  4a26b739826a2a88d15f93f7ddcd3fe6eeaa083fc575",
      "0x82fa00c0e93b077e6e8b07e247f2d72c5a3825ba6b94105659a99798723acf14e6c913475a330c3293aa
      ↪  cf4b8e0fbd1792f8c50e1093c1817c1e4645ec378e08",
      "0xb8ece486bea9ed49f50b07c9f0e2c66cde7b20d0d0fbdad9536cf7525c47f7da76b50b682c6fac8f7686
      ↪  7fc1617c0070eada45cd70a2bee7e2cb6718728491a4",
      "0xc1e0d66bbcc96967f3c94ed08e05528ecf50b7cc06186b2a8b0973b7744a1fe46786bd68891d28022471
      ↪  1c58fc51edb322f12b899a3ad913b8230bdf4d2a28d0",
      "0x91653adb04e71881cea254bd5cf2b13cd11f00ff459919b4b63a1ccad36bf767039257690605f1200438
      ↪  e52085f97eae6ab6385ec4141f230f63b788fff90b0e",
      ...
    ],
    "IP_address": [
      "0x1b483e6acb2750d9dbc8583d4241833ba81d5ba6c43a43ff8bc3eecb490ceb9c3077d0d91e9458d33c1a
      ↪  8f8c37de39e02582b13981ff022551f97d99dcd2898c",
      "0xc92083ccc56bde0c405d312d830d06394c1f12d927872a610368470e26f1eaab74db766aae1c84efa501
      ↪  e64d9b98d3ba5173f48fc6adf316a78f3d2ea863370c",
      "0xd292541de4b426e5615bbf96e9f5920674ecf6e07a222e3328f0873b1b354376705a4bbe2d9eb67157dd
      ↪  6dd745b1444b5ce4f4bbabee27cd8884d6af18d88673",
      "0x402dacc142528a10cd0a4a62780b230358e099846966d5b67270be56ee3f270c0ba2a8500248b647ccec
      ↪  91d202e8c1d15ee7764ada0616fc28360954756f7ab0",
      "0xe1268b8e5f7808f5fee9e9b428bcd063f957f4b682ec4855e93fca9c552704c5c036d158129ac6262201
      ↪  631ff48455ad76eb9586f7d689577e3ed7464679ad16",
      "0xfd365f51852d4d5153a12d71522d5125f1eef3946c7a2478b143cecd1201f2d68f04b3ba096eca8e11c8
      ↪  064e6f0950ae58290b81f103cb65c5b8a0284c31375e",
      ...
    ],
    "IP_address": [
      "0xf2ec35ab477de705ca25280c6149920e0d040b1ef788700a8e43c20750588c40b7e4927c75b8c896c75a
      ↪  f45ab1ed664501df454c3324ac2facabb82fda3dd4a3",
      "0xbd3345d9040d5ead70d837ae2db3172d59431a75c3abdd998343af49ccd4af4f212f454d8232ff85d187
      ↪  4b4683240de9096494b97ec90555f1975d3890331160",
      "0xaf51efdea75c19658355a2e6b62abb37be00fa63a5d6e4d7b9769323f240118bcafc9c96ac731c8dac6a
      ↪  9632ef93afcad951805b47169676c3003c20e597cae5",
      "0xc664b659aed4533f23f812bdbb92a7456658bc6dab787629c9f83847d0efaf6e0af91900f28b2b6b15f9
      ↪  f00d767f21c419f995f7e370500799445a9f9ac45e67",
      "0x3ae65f7332c2e1d013da878fb2d2d608619ef2c0942629ea70af16a098cade7050f2df2b6ed1991f4154
      ↪  41365c9d0a1101b3261c6bd2efa7bcc59b3d132fff47",
      "0x17cf3b9443037df764820d88971c8b8a7d004b080fc11ee9c57b2ffd2c45281a6426eb0b5ad67a4a2f8a
      ↪  d171f636d56dba83b445025d5acd68c74353dc3a4efc",
      ...
    ],
    "IP_address": [
      "0x14a4ad83412cc763f5f8bc96c5c33e5dd591a2b514c70262539682803831cf3c22508761f023af5392fc
      ↪  7a17512b1952242867767b6e8ce16f526e02d27314c8",
      "0xa978517aeae13642fa8fe030e525897b05d820d2f1ee7df53bae5e8fa65fe20f2d112809b0daf0cfb466
      ↪  0a4d25b3310839e66d9b521e3c5a599c5218aaa5467d",
      "0x2cc6f7e0be5b0b45cacc72bcddca89e6bf8845649d174b41a91ff0526b0727fea06458a51c1bd6fe8b14
      ↪  e21c9181c5d006341157c476a590f699d8225a9797cc",
      "0xf54cace1135e5a98c8d2b07f4f05fb1edc997ec6f948d87419fc066324dc6284d79e4240904fd9b1282d
      ↪  9a15c026004ef543029a8c0aa5833c3519eb5f6b7928",
    ]
    ...
  }
}
```

Fichier *groups_ date.json* contenant les préfixes de NODEID d'une taille donnée regroupant 3 nœuds ou plus (ici la taille de préfixe maximum est fixée à 18 bits) :


```
{
  "threshold": 18,
  "results": {
    "0x31ae": 3,
    "0x5137": 3,
    "0x66dd": 3,
    "0x6c75": 3,
    "0x6ef5": 3,
    "0x7f8e": 3,
    "0x8648": 3,
    "0x883e": 3,
    "0xa5d3": 3,
    "0xabff": 3,
    "0xbaab": 3,
    "0xc13a": 3,
    "0xcdeb": 3,
    "0xd100": 3,
    "0xd2e1": 3,
    "0xe2be": 3,
    "0xe72e": 3,
    "0xf405": 3,
    "0x109c8": 3,
    "0x12141": 3,
    "0x131c1": 3,
    "0x1364d": 3,
    "0x14a42": 3,
    "0x14b9b": 3,
    "0x15e06": 3,
    "0x15e2c": 3,
    "0x16a2c": 3,
    "0x16eab": 3,
    "0x171ea": 3,
    "0x184eb": 3,
    "0x185b8": 3,
    "0x1a025": 3,
    "0x1ac3a": 3,
    "0x1b13c": 3,
    "0x1d27d": 3,
    "0x1e476": 3,
    "0x1e539": 3,
    "0x1e8e6": 3,
    "0x1f194": 3,
    "0x1fafd": 3,
    "0x207eb": 3,
    "0x22614": 3,
    "0x22ea9": 3,
    "0x23435": 3,
    "0x23545": 3,
    "0x26023": 3,
    "0x264e8": 3,
    "0x265e1": 3,
    "0x26b7d": 3,
    "0x26bc2": 3,
    "0x28189": 3,
    "0x286a9": 3,
    "0x2899e": 3,
```

```
    "0x29b3c": 3,
    "0x2a7c2": 3,
    "0x2b337": 3,
    "0x2b384": 3,
    "0x2b4ae": 3,
    "0x2bd75": 3,
    "0x2c205": 3,
    "0x2c545": 3,
    "0x2d092": 3,
    "0x2e112": 3,
    "0x2e36a": 4,
    "0x2eb56": 4,
    "0x2f4c7": 3,
    "0x307d1": 3,
    "0x30a56": 3,
    "0x31d57": 3,
    "0x32d96": 3,
    "0x34d31": 4,
    "0x354dd": 3,
    "0x355d6": 3,
    "0x35fb4": 3,
    "0x36192": 3,
    "0x3678a": 3,
    "0x3735e": 3,
    "0x3754a": 3,
    "0x381c7": 3,
    "0x3c039": 3,
    "0x3c0f9": 3,
    "0x3d45f": 3,
    "0x3f1ef": 3,
    "0x3f4e3": 3
  }
}
```

Fichier *subnets_ date.json* contenant les adresses IP (et ports) des pairs regroupées au sein d'un même réseau /24 dont la taille est supérieur à un seuil (ici 24) :


```
{
  "threshold": 24,
  "results": {
    "IP_address/24": {
      "5": [30303],
      "8": [30303],
      "9": [30303],
      "10": [30303],
      "12": [30303],
      "14": [30303],
      "15": [30303],
      "17": [30303],
      "18": [30303],
      "21": [30303],
      "22": [30303],
      "23": [30303],
      "24": [30303],
      "25": [30303],
      "26": [30303],
      "27": [30303],
      "28": [30303],
      "30": [30303],
      "31": [30303],
      "32": [30303],
      "33": [30303],
      "34": [30303],
      "37": [30303],
      "39": [30303],
      "40": [30303],
      "41": [30303],
      "42": [30303],
      "44": [30303],
      "45": [30303],
      "46": [30303],
      "47": [30303],
      "49": [30303],
      "50": [30303]
    },
    "IP_address/24": {
      "131": [30303],
      "132": [30303],
      "133": [30303],
      "134": [30303],
      "135": [30303],
      "136": [30303],
      "137": [30303],
      "138": [30303],
      "139": [30303],
      "140": [30303],
      "141": [30303],
      "142": [30303],
      "143": [30303],
      "144": [30303],
      "148": [30303],
      "150": [30303],
      "151": [30303],
```

```
      "152": [30303],
      "156": [30303],
      "158": [30303],
      "159": [30303],
      "163": [30303],
      "164": [30303],
      "166": [30303],
      "168": [30303],
      "172": [30303],
      "176": [30303],
      "181": [30303],
      "193": [30303],
      "197": [30303],
      "200": [30303],
      "203": [30303]
    }
    ...
  }
}
```

Bibliographie

[AKR+13] Elli Androulaki, Ghassan O. Karame, Marc Roeschlin, Tobias Scherer et Srdjan Capkun : Evaluating user privacy in bitcoin. *In* Ahmad-Reza Sadeghi, éditeur : *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 34–51, Berlin, Heidelberg, 2013. Springer.

[AMMV18] Maria Apostolaki, Gian Marti, Jan Müller et Laurent Vanbever : Sabre : Protecting bitcoin against routing attacks. `http://arxiv.org/abs/1808.06254`, août 2018.

[AMRS08] Luca Maria Aiello, Marco Milanesio, Giancarlo Ruffo et Rossano Schifanella : Tempering kademlia with a robust identity based system. *In 2008 Eighth International Conference on Peer-to-Peer Computing*, pages 30–39, septembre 2008.

[AZV17] Maria Apostolaki, Aviv Zohar et Laurent Vanbever : Hijacking bitcoin : Routing attacks on cryptocurrencies. *In 2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392, San Jose, CA, USA, mai 2017. IEEE. `http://ieeexplore.ieee.org/document/7958588/`.

[BB06] René Brunner et E. Biersack : A performance evaluation of the kad-protocol. *Mémoire de Master, Université de Mannheim, Allemagne et Institut Eurecom, France.*, 2006.

[BFV19] Mirko Bez, Giacomo Fornari et Tullio Vardanega : The scalability challenge of ethereum : An initial quantitative analysis. *In 2019 IEEE International Conference on Service-Oriented System Engineering (SOSE)*, pages 167–176, avril 2019.

[BKP14] Alex Biryukov, Dmitry Khovratovich et Ivan Pustogarov : Deanonymisation of clients in bitcoin p2p network. *In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 15–29, Scottsdale, Arizona, USA, novembre 2014. Association for Computing Machinery. `https://doi.org/10.1145/2660267.2660379`.

[BNM+14] Joseph Bonneau, Arvind Narayanan, Andrew Miller, Jeremy Clark, Joshua A. Kroll et Edward W. Felten : Mixcoin : Anonymity for bitcoin with accountable mixes. *In* Nicolas Christin et Reihaneh Safavi-Naini, éditeurs : *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 486–504, Berlin, Heidelberg, 2014. Springer.

[BOLL14] George Bissias, A. Pinar Ozisik, Brian N. Levine et Marc Liberatore : Sybil-resistant mixing for bitcoin. *In Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 149–158, New York, NY, USA, novembre 2014. Association for Computing Machinery.

[But14] Vitalik Buterin : A next-generation smart contract and decentralized application platform. *Ethereum project white paper*, 3(37):2–1, 2014.

[But16] Vitalik Buterin : Dao exploit. https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/, juin 2016.

[CBR+22] Rajasekhar Chaganti, Rajendra V. Boppana, Vinayakumar Ravi, Kashif Munir, Mubarak Almutairi, Furqan Rustam, Ernesto Lee et Imran Ashraf : A comprehensive review of denial of service attacks in blockchain ecosystem and open challenges. *IEEE Access*, 10:96538–96555, 2022.

[CCF09] Thibault Cholez, Isabelle Chrisment et Olivier Festor : Evaluation of sybil attacks protection schemes in kad. *In* Ramin Sadre et Aiko Pras, éditeurs : *Scalability of Networks and Services*, Lecture Notes in Computer Science, pages 70–82, Berlin, Heidelberg, 2009. Springer.

[CCF10] Thibault Cholez, Isabelle Chrisment et Olivier Festor : Monitoring and controlling content access in kad. *In International Conference on Communications - ICC 2010*, mai 2010. https://hal.inria.fr/inria-00490347.

[CCFD13] Thibault Cholez, Isabelle Chrisment, Olivier Festor et Guillaume Doyen : Detection and mitigation of localized attacks in a widely deployed p2p network. *Peer-to-Peer Netw. Appl.*, 6(2):155–174, juin 2013. https://doi.org/10.1007/s12083-012-0137-7.

[CDG+03] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron et Dan S. Wallach : Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314, décembre 2003. https://doi.org/10.1145/844128.844156.

[Cho11] Thibault Cholez : *Supervision Des Réseaux Pair à Pair Structurés Appliquée à La Sécurité Des Contenus*. Thèse de doctorat, Université Nancy 1, juin 2011. http://theses.fr/2011NAN10036.

[CKS+06] Tyson Condie, Varun Kacholia, Sriram Sank, Joseph M. Hellerstein et Petros Maniatis : Induced churn as shelter from routing-table poisoning. *In Proceedings of the Network and Distributed System Security Symposium, NDSS 2006, San Diego, California, USA*, 2006. https://www.ndss-symposium.org/ndss2006/induced-churn-shelter-routing-table-poisoning/.

[CMVM18] Anamika Chauhan, Om Prakash Malviya, Madhav Verma et Tejinder Singh Mor : Blockchain and scalability. *In 2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 122–128, juillet 2018.

[CSKLR18] Mauro Conti, E. Sandeep Kumar, Chhagan Lal et Sushmita Ruj : A survey on security and privacy issues of bitcoin. *IEEE Commun. Surv. Tutorials*, 20(4):3416–3452, 2018. https://ieeexplore.ieee.org/document/8369416/.

[DBG18] Varun Deshpande, Hakim Badis et Laurent George : Btcmap : Mapping bitcoin peer-to-peer network topology. *In 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks (PEMWN)*, pages 1–6, Toulouse, septembre 2018. IEEE. https://ieeexplore.ieee.org/document/8548904/.

[DBP+19] Sergi Delgado-Segura, Surya Bakshi, Cristina Pérez-Solà, James Litton, Andrew Pachulski, Andrew Miller et Bobby Bhattacharjee : Txprobe : Discovering bitcoin's network topology using orphan transactions. *In* Ian Goldberg et Tyler Moore, éditeurs : *Financial Cryptography and Data Security*, Lecture

Notes in Computer Science, pages 550–566, Cham, 2019. Springer International Publishing.

[DD19] Richard Dennis et Jules Pagna Disso : An analysis into the scalability of bitcoin and ethereum. *In* Xin-She Yang, Simon Sherratt, Nilanjan Dey et Amit Joshi, éditeurs : *Third International Congress on Information and Communication Technology*, Advances in Intelligent Systems and Computing, pages 619–627, Singapore, 2019. Springer.

[DH06] J. Dinger et H. Hartenstein : Defending the sybil attack in p2p networks : Taxonomy, challenges, and a proposal for self-registration. *In First International Conference on Availability, Reliability and Security (ARES'06)*, pages 8 pp.–763, avril 2006.

[DLKA05] George Danezis, Chris Lesniewski-Laas, M. Frans Kaashoek et Ross Anderson : Sybil-resistant dht routing. *In* Sabrina de Capitani di Vimercati, Paul Syverson et Dieter Gollmann, éditeurs : *Computer Security – ESORICS 2005*, Lecture Notes in Computer Science, pages 305–318, Berlin, Heidelberg, 2005. Springer.

[DN93] Cynthia Dwork et Moni Naor : Pricing via processing or combatting junk mail. *In* Ernest F. Brickell, éditeur : *Advances in Cryptology — CRYPTO' 92*, Lecture Notes in Computer Science, pages 139–147, Berlin, Heidelberg, 1993. Springer.

[Dou02] John R. Douceur : The sybil attack. *In* Peter Druschel, Frans Kaashoek et Antony Rowstron, éditeurs : *Peer-to-Peer Systems*, Lecture Notes in Computer Science, pages 251–260, Berlin, Heidelberg, 2002. Springer.

[DW13] Christian Decker et Roger Wattenhofer : Information propagation in the bitcoin network. *In IEEE P2P 2013 Proceedings*, pages 1–10, Trento, Italy, septembre 2013. IEEE. http://ieeexplore.ieee.org/document/6688704/.

[ECP21a] Jean-Philippe Eisenbarth, Thibault Cholez et Olivier Perrin : A comprehensive study of the bitcoin p2p network. *In 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 105–112, septembre 2021. https://hal.inria.fr/hal-03380595.

[ECP21b] Jean-Philippe Eisenbarth, Thibault Cholez et Olivier Perrin : An open measurement dataset on the bitcoin p2p network. *In 2021 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, mai 2021.

[ECP22] Jean-Philippe Eisenbarth, Thibault Cholez et Olivier Perrin : Ethereum's peer-to-peer network monitoring and sybil attack prevention. *J Netw Syst Manage*, 30(4):65, octobre 2022. https://link.springer.com/10.1007/s10922-022-09676-2.

[Eisa] Jean-Philippe Eisenbarth : Crawleth. https://gitlab.inria.fr/jeisenba/Crawleth.

[Eisb] Jean-Philippe Eisenbarth : Github jpeisenbarth/bitnodes forked from ayeowch/-bitnodes. https://github.com/jpeisenbarth/bitnodes/tree/add_statistics.

[Eisc] Jean-Philippe Eisenbarth : Sybil-prevention. https://gitlab.inria.fr/jeisenba/sybil-prevention.

[Eis21] Jean-Philippe Eisenbarth : Ethereum p2p network study, dataset overview. https://concordia-eth-p2p.lhs.loria.fr/, 2021.

[Eis22] Jean-Philippe Eisenbarth : Jpeisenbarth/go-ethereum. `https://github.com/jpeisenbarth/go-ethereum`, mai 2022.

[EPJ19] Meryam Essaid, Sejin Park et Hongteak Ju : Visualising bitcoin's dynamic p2p network topoogy and performance. *In 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 141–145, Seoul, Korea (South), mai 2019. IEEE. `https://ieeexplore.ieee.org/document/8751305/`.

[EPJ20] Meryam Essaid, Sejin Park et Hong-Taek Ju : Bitcoin's dynamic peer-to-peer topology. *Int J Network Mgmt*, 30(5), septembre 2020. `https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.2106`.

[ES14] Ittay Eyal et Emin Gün Sirer : Majority is not enough : Bitcoin mining is vulnerable. *In* Nicolas Christin et Reihaneh Safavi-Naini, éditeurs : *Financial Cryptography and Data Security*, volume 8437, pages 436–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. `https://doi.org/10.1007/978-3-662-45472-5_28`.

[Etha] Ethernodes : The ethereum network & node explorer. `https://ethernodes.org/`.

[Ethb] Etherscan : Ethereum full node sync (default) chart | etherscan. `http://etherscan.io/chartsync/chaindefault`.

[Ethc] Etherscan : Ethereum node tracker. `http://etherscan.io/nodetracker`.

[FMR+09] R. Fantacci, L. Maccari, M. Rosi, L. Chisci, L. M. Aiello et M. Milanesio : Avoiding eclipse attacks on kad/kademlia : An identity based approach. *In 2009 IEEE International Conference on Communications*, pages 1–5, juin 2009.

[Fou20] The Ethereum Foundation : Ask about geth : Snapshot acceleration. `https://blog.ethereum.org/2020/07/17/ask-about-geth-snapshot-acceleration`, juillet 2020.

[FSW14] Sebastian Feld, Mirco Schönfeld et Martin Werner : Analyzing the deployment of bitcoin's p2p network under an as-level perspective. *Procedia Computer Science*, 32:1121–1126, 2014. `https://linkinghub.elsevier.com/retrieve/pii/S187705091400742X`.

[GBE+18] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse et Emin Gün Sirer : Decentralization in bitcoin and ethereum networks. *In* Sarah Meiklejohn et Kazue Sako, éditeurs : *Financial Cryptography and Data Security*, volume 10957, pages 439–457. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. `http://link.springer.com/10.1007/978-3-662-58387-6_24`.

[GKW+16] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf et Srdjan Capkun : On the security and performance of proof of work blockchains. *In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security - CCS'16*, pages 3–16, Vienna, Austria, 2016. ACM Press. `http://dl.acm.org/citation.cfm?doid=2976749.2978341`.

[GNH18] Matthias Grundmann, Till Neudecker et Hannes Hartenstein : Exploiting transaction accumulation and double spends for topology inference in bitcoin. *In* Aviv Zohar, Ittay Eyal, Vanessa Teague, Jeremy Clark, Andrea Bracciali, Federico Pintore et Massimiliano Sala, éditeurs : *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 113–126, Berlin, Heidelberg, 2018. Springer.

[GP18a] Cyril Grunspan et Ricardo Pérez-Marco : On profitability of selfish mining. *arXiv :1805.08281 [cs, math]*, mai 2018. http://arxiv.org/abs/1805.08281.

[GP18b] Cyril Grunspan et Ricardo Pérez-Marco : On profitability of stubborn mining. *arXiv :1808.01041 [cs, math]*, août 2018. http://arxiv.org/abs/1808.01041.

[GP18c] Cyril Grunspan et Ricardo Pérez-Marco : On profitability of trailing mining. *arXiv :1811.09322 [cs, math]*, novembre 2018. http://arxiv.org/abs/1811.09322.

[Gra] Vincent Gramoli : Blockchain scalability and its foundations in distributed systems. https://www.coursera.org/learn/blockchain-scalability.

[Gra20] Vincent Gramoli : From blockchain consensus back to byzantine consensus. *Future Generation Computer Systems*, 107:760–769, juin 2020. https://www.sciencedirect.com/science/article/pii/S0167739X17320095.

[GSW+19] Y. Gao, J. Shi, X. Wang, Q. Tan, C. Zhao et Z. Yin : Topology measurement and analysis on ethereum p2p network. *In 2019 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7, juin 2019.

[HKZG15] Ethan Heilman, Alison Kendler, Aviv Zohar et Sharon Goldberg : Eclipse attacks on bitcoin's peer-to-peer network. *In Proceedings of the 24th USENIX Conference on Security Symposium*, SEC'15, pages 129–144, Berkeley, CA, USA, 2015. USENIX Association.

[IM21] Mubashar Iqbal et Raimundas Matulevičius : Exploring sybil and double-spending risks in blockchain systems. *IEEE Access*, 9:76153–76177, 2021.

[Int18] Internet Corporation for Assigned Names and Numbers : Recommendations on anonymization processes for source ip addresses submitted for future analysis. https://www.icann.org/en/system/files/files/rssac-040-07aug18-en.pdf, août 2018.

[ISTY19] Muhammad Anas Imtiaz, David Starobinski, Ari Trachtenberg et Nabeel Younis : Churn in the bitcoin network : Characterization and impact. *In 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 431–439, Seoul, Korea (South), mai 2019. IEEE. https://ieeexplore.ieee.org/document/8751297/.

[JJ99] Markus Jakobsson et Ari Juels : Proofs of work and bread pudding protocols(extended abstract). *In* Bart Preneel, éditeur : *Secure Information Networks : Communications and Multimedia Security IFIP TC6/TC11 Joint Working Conference on Communications and Multimedia Security (CMS'99) September 20–21, 1999, Leuven, Belgium*, IFIP — The International Federation for Information Processing, pages 258–272. Springer US, Boston, MA, 1999. https://doi.org/10.1007/978-0-387-35568-9_18.

[KAC12] Ghassan O. Karame, Elli Androulaki et Srdjan Capkun : Double-spending fast payments in bitcoin. *In Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 906–917, New York, NY, USA, octobre 2012. Association for Computing Machinery. https://doi.org/10.1145/2382196.2382292.

[KLR09] Michael Kohnen, Mike Leske et Erwin P. Rathgeb : Conducting and optimizing eclipse attacks in the kad peer-to-peer network. *In* Luigi Fratta, Henning Schulzrinne, Yutaka Takahashi et Otto Spaniol, éditeurs : *NETWORKING*

2009, Lecture Notes in Computer Science, pages 104–116, Berlin, Heidelberg, 2009. Springer.

[KMM+18] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller et Michael Bailey : Measuring ethereum network peers. *In Proceedings of the Internet Measurement Conference 2018 on - IMC '18*, pages 91–104, Boston, MA, USA, 2018. ACM Press.

[KP19] Aggelos Kiayias et Giorgos Panagiotakos : On trees, chains and fast transactions in the blockchain. *In* Tanja Lange et Orr Dunkelman, éditeurs : *Progress in Cryptology – LATINCRYPT 2017*, volume 11368, pages 327–351. Springer International Publishing, Cham, 2019. http://link.springer.com/10.1007/978-3-030-25283-0_18.

[KSE21] Periklis Kostamis, Andreas Sendros et Pavlos Efraimidis : Exploring ethereum's data stores : A cost and performance comparison. *In 2021 3rd Conference on Blockchain Research Applications for Innovative Networks and Services (BRAINS)*, pages 53–60, septembre 2021.

[LJC+17] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo et Qiaoyan Wen : A survey on the security of blockchain systems. *Future Generation Computer Systems*, page S0167739X17318332, août 2017. https://linkinghub.elsevier.com/retrieve/pii/S0167739X17318332.

[LMT09] Francois Lesueur, Ludovic Me et Valerie Viet Triem Tong : An efficient distributed pki for structured p2p networks. *In 2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 1–10, septembre 2009.

[LMVTT08] François Lesueur, Ludovic Mé et Valérie Viet Triem Tong : Detecting and excluding misbehaving nodes in a p2p network. *In 8th International Conference on Innovative Internet Community Systems (I2CS)*, pages –, Martinique, juin 2008. https://hal-supelec.archives-ouvertes.fr/hal-00353016.

[LNR06] J. Liang, N. Naoumov et K. W. Ross : The index poisoning attack in p2p file sharing systems. *In Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications, 2006. https://www.infona.pl//resource/bwmeta1.element.ieee-art-000004146885.

[LSM06] Brian Neil Levine, Clay Shields et N. Boris Margolin : A survey of solutions to the sybil attack. Rapport technique, University of Massachusetts Amherst, octobre 2006. http://prisms.cs.umass.edu/brian/pubs/levine.sybil.tr.2006.pdf.

[Max] MaxMind : Geolite2 free geolocation data, maxmind developer portal. https://dev.maxmind.com/geoip/geolite2-free-geolocation-data?lang=en.

[MEJ20] Soo Hoon Maeng, Meryam Essaid et Hong Taek Ju : Analysis of ethereum network properties and behavior of influential nodes. *In 2020 21st Asia-Pacific Network Operations and Management Symposium (APNOMS)*, pages 203–207, septembre 2020.

[MHG18] Yuval Marcus, Ethan Heilman et Sharon Goldberg : Low-resource eclipse attacks on ethereum's peer-to-peer network. *IACR Cryptology ePrint Archive*, 2018: 236, 2018.

[Mil13] Andrew Miller : Feather-forks : Enforcing a blacklist with sub-50% hash power. https://bitcointalk.org/index.php?topic=312668.0, 2013.

[MLP+15] Andrew Miller, James Litton, Andrew Pachulski, Neal Gupta, Dave Levin, Neil Spring et Bobby Bhattacharjee : Discovering bitcoin's public topology and influential nodes. 2015.

[MM02] Petar Maymounkov et David Mazières : Kademlia : A peer-to-peer information system based on the xor metric. *In* Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Peter Druschel, Frans Kaashoek et Antony Rowstron, éditeurs : *Peer-to-Peer Systems*, volume 2429, pages 53–65. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[MPJ+13] Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker et Stefan Savage : A fistful of bitcoins : Characterizing payments among men with no names. *In Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, pages 127–140, New York, NY, USA, octobre 2013. Association for Computing Machinery. https://doi.org/10.1145/2504730.2504747.

[NAH16] T. Neudecker, P. Andelfinger et H. Hartenstein : Timing analysis for inferring the topology of the bitcoin peer-to-peer network. *In 2016 Intl IEEE Conferences on Ubiquitous Intelligence Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*, pages 358–367, juillet 2016.

[Nak08] Satoshi Nakamoto : Bitcoin : A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

[NEIP18] Hoang-Long Nguyen, Jean-Philippe Eisenbarth, Claudia-Lavinia Ignat et Olivier Perrin : Blockchain-based auditing of transparent log servers. *In* Florian Kerschbaum et Stefano Paraboschi, éditeurs : *Data and Applications Security and Privacy XXXII*, Lecture Notes in Computer Science, pages 21–37, Cham, 2018. Springer International Publishing.

[Neu19] Till Neudecker : *Characterization of the Bitcoin Peer-to-Peer Network (2015-2018).* 2019.

[NG17] Christopher Natoli et Vincent Gramoli : The balance attack or why forkable blockchains are ill-suited for consortium. *In 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 579–590, Denver, CO, USA, juin 2017. IEEE. http://ieeexplore.ieee.org/document/8023156/.

[NIS] NIST : Nvd - cvss v3 calculator. https://nvd.nist.gov/vuln-metrics/cvss/v3-calculator.

[NKMS16] Kartik Nayak, Srijan Kumar, Andrew Miller et Elaine Shi : Stubborn mining : Generalizing selfish mining and combining with an eclipse attack. *In 2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 305–320, Saarbrucken, mars 2016. IEEE.

[Ope] OpenEthereum DAO : Openethereum. https://github.com/openethereum/openethereum.

[PDBU21] P. Poornima Devi, Srinivasan Ananthanarayanan Bragadeesh et Arumugam Umamakeswari : Secure data management using ipfs and ethereum. *In* Valentina E. Balas, Aboul Ella Hassanien, Satyajit Chakrabarti et Lopa Mandal,

éditeurs : *Proceedings of International Conference on Computational Intelligence, Data Science and Cloud Computing*, Lecture Notes on Data Engineering and Communications Technologies, pages 565–578, Singapore, 2021. Springer.

[PDMCA18] Giuseppe Pappalardo, Tiziana Di Matteo, Guido Caldarelli et Tomaso Aste : Blockchain inefficiency in the bitcoin peers network. *EPJ Data Sci.*, 7(1):30, décembre 2018. https://epjdatascience.springeropen.com/articles/10.1140/epjds/s13688-018-0159-3.

[Qui19] Mikerah Quintyne-Collins : Short paper : Towards characterizing sybil attacks in cryptocurrency mixers. https://eprint.iacr.org/2019/1111, 2019.

[REMLP07] H. Rowaihy, W. Enck, P. McDaniel et T. La Porta : Limiting sybil attacks in structured p2p networks. *In IEEE INFOCOM 2007 - 26th IEEE International Conference on Computer Communications*, pages 2596–2600, mai 2007.

[RS13] Dorit Ron et Adi Shamir : Quantitative analysis of the full bitcoin transaction graph. *In* Ahmad-Reza Sadeghi, éditeur : *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 6–24, Berlin, Heidelberg, 2013. Springer.

[RSK+20] Bruno Rodrigues, Eder Scheid, Christian Killer, Muriel Franco et Burkhard Stiller : Blockchain signaling system (bloss) : Cooperative signaling of distributed denial-of-service attacks. *J Netw Syst Manage*, 28(4):953–989, octobre 2020. https://doi.org/10.1007/s10922-020-09559-4.

[SCDR04] Atul Singh, Miguel Castro, Peter Druschel et Antony Rowstron : Defending against eclipse attacks on overlay networks. *In Proceedings of the 11th Workshop on ACM SIGOPS European Workshop*, EW 11, pages 21–es, New York, NY, USA, septembre 2004. Association for Computing Machinery. https://doi.org/10.1145/1133572.1133613.

[SEB07] Moritz Steiner, Taoufik En-Najjary et Ernst W. Biersack : Exploiting kad : Possible uses and misuses. *SIGCOMM Comput. Commun. Rev.*, 37(5):65–70, octobre 2007. https://doi.org/10.1145/1290168.1290176.

[SM19] Sarwar Sayeed et Hector Marco-Gisbert : Assessing blockchain consensus and security mechanisms against the 51% attack. *Applied Sciences*, 9(9):1788, janvier 2019. https://www.mdpi.com/2076-3417/9/9/1788.

[SMP19] P. Swathi, C. Modi et D. Patel : Preventing sybil attack in blockchain using distributed behavior monitoring of miners. *In 2019 10th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–6, juillet 2019.

[SN18] Amool Sudhan et Manisha J Nene : Peer selection techniques for enhanced transaction propagation in bitcoin peer-to-peer network. *In 2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 679–684, juin 2018.

[SNDW06] A. Singh, T.-W. Ngan, P. Druschel et D. S. Wallach : Eclipse attacks on overlay networks : Threats and defenses. *In Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, pages 1–12, avril 2006.

[SZ15] Yonatan Sompolinsky et Aviv Zohar : Secure high-rate transaction processing in bitcoin. *In* Rainer Böhme et Tatsuaki Okamoto, éditeurs : *Financial Cryp-*

tography and Data Security, Lecture Notes in Computer Science, pages 507–527, Berlin, Heidelberg, 2015. Springer.

[TCM+20] Muoi Tran, Inho Choi, Gi Jun Moon, Anh V Vu et Min Suk Kang : A stealthier partitioning attack against bitcoin peer-to-peer network. *In 2020 IEEE Symposium on Security and Privacy (SP)*, pages 496–511, 2020.

[Thea] The bcoin developers : Github bcoin-org/bcoin. https://github.com/bcoin-org/bcoin.

[Theb] The Bitcoin community : Cache responses to getaddr to prevent topology leaks by naumenkogs · pull request #18991 · bitcoin/bitcoin. https://github.com/bitcoin/bitcoin/pull/18991.

[Thec] The Bitcoin community : Satoshi client node discovery. https://en.bitcoin.it/wiki/Satoshi_Client_Node_Discovery.

[Thed] The Bitcoin Core developers : Github bitcoin/bitcoin. Bitcoin. https://github.com/bitcoin/bitcoin.

[Thee] The Bitcoin Core developers : Reduce fingerprinting through timestamps in 'addr' messages. · bitcoin/bitcoin@9c27379. https://github.com/bitcoin/bitcoin/commit/9c2737901b5203f267d21d728019d64b46f1d9f3.

[Thef] The Bitcoin Core developers : Releases of bitcoin/bitcoin. https://github.com/bitcoin/bitcoin/releases.

[Theg] The bitcoinj developers : Github bitcoinj/bitcoinj. bitcoinj. https://github.com/bitcoinj/bitcoinj.

[Theh] The btcsuite developers : Github btcsuite/btcd. Bitcoin in Go. https://github.com/btcsuite/btcd.

[Thei] The Gnutella developers : Gnutella - stable - 0.4. https://rfc-gnutella.sourceforge.net/developer/stable/index.html.

[Thej] The Go-Ethereum developers : Ethereum development documentation. https://ethereum.org.

[Thek] The go-ethereum developers : Ethereum/go-ethereum. https://github.com/ethereum/go-ethereum.

[Thel] The go-ethereum developers : Ethereum/go-ethereum, pull request #2740 (firescar96) : Add ability to remove peers via admin interface. https://github.com/ethereum/go-ethereum/pull/2740.

[Them] The Go-Ethereum developers : Home · ethereum/wiki wiki. https://github.com/ethereum/wiki.

[The21] The Ethereum Foundation : Devp2p - ethereum peer-to-peer networking specifications. https://github.com/ethereum/devp2p, 2021.

[The22] The Go-Ethereum developers : Ethereum/go-ethereum. ethereum, octobre 2022. https://github.com/ethereum/go-ethereum.

[Tho16] Lee Thomas : Blockchainillustrations/ethereum at master · 4c656554/blockchainillustrations. https://github.com/4c656554/BlockchainIllustrations, 2016.

[UPS11] Guido Urdaneta, Guillaume Pierre et Maarten Van Steen : A survey of dht security techniques. *ACM Comput. Surv.*, 43(2):8 :1–8 :49, février 2011. https://doi.org/10.1145/1883612.1883615.

[WG16] Karl Wüst et Arthur Gervais : Ethereum eclipse attacks. Rapport technique, ETH Zurich, 2016. http://hdl.handle.net/20.500.11850/121310.

[Woo14] Gavin Wood : Ethereum : A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[WTC+08] Peng Wang, James Tyra, Eric Chan-Tin, Tyson Malchow, Denis Foo Kune, Nicholas Hopper et Yongdae Kim : Attacking the kad network. *In Proceedings of the 4th International Conference on Security and Privacy in Communication Netowrks*, SecureComm '08, pages 1–10, New York, NY, USA, septembre 2008. Association for Computing Machinery. https://doi.org/10.1145/1460877.1460907.

[WZR21] Maximilian Wöhrer, Uwe Zdun et Stefanie Rinderle-Ma : Architecture design of blockchain-based applications. *In 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, septembre 2021.

[XGS+20] Guangquan Xu, Bingjiang Guo, Chunhua Su, Xi Zheng, Kaitai Liang, Duncan S. Wong et Hao Wang : Am i eclipsed? a smart detector of eclipse attacks for ethereum. *Computers & Security*, 88:101604, janvier 2020.

[YGKX08] Haifeng Yu, Phillip B. Gibbons, Michael Kaminsky et Feng Xiao : Sybillimit : A near-optimal social network defense against sybil attacks. *In 2008 IEEE Symposium on Security and Privacy (Sp 2008)*, pages 3–17, mai 2008.

[YKGF06] Haifeng Yu, Michael Kaminsky, Phillip B. Gibbons et Abraham Flaxman : Sybilguard : Defending against sybil attacks via social networks. *In Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '06, pages 267–278, New York, NY, USA, août 2006. Association for Computing Machinery. https://doi.org/10.1145/1159913.1159945.

[ZL19] Shijie Zhang et Jong-Hyouk Lee : Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE Transactions on Industrial Informatics*, 15(10):5715–5722, octobre 2019.

[ZP19] Ren Zhang et Bart Preneel : Lay down the common metrics : Evaluating proof-of-work consensus protocols' security. *In 2019 IEEE Symposium on Security and Privacy (SP)*, pages 175–192, mai 2019.

Abstract

Analysis, exploitation and protection of public blockchains' peer-to-peer networks

Blockchains rely on P2P networks that are essential to their proper functioning, as they ensure the dissemination of transactions and blocks to all parties. While Bitcoin and Ethereum – the two main public blockchains – are now worth trillions of dollars, attracting new users every day, few studies focus on the network aspects, although the literature shows that many problems can reduce the reliability of public P2P networks.

In this thesis, we first focused on the monitoring of the P2P networks of the Bitcoin and Ethereum blockchains. We implemented a crawler for each network able to discover all connected peers and analyzed data from several months of measurement campaigns. Different criteria that can affect the reliability of the network were studied, such as the number of peers, their geographical distribution, their distribution over the IP network, the churn, the proportion of clients with known vulnerabilities, the existence of daily connection patterns or the ability to infer topology. It appears that both networks show good properties on all these points.

Starting from the observation that, on the one hand, the Ethereum P2P network based on a distributed hash table (DHT) is largely unexploited, as no data is stored in the DHT, and on the other hand, the storage of the blockchain data is only growing (which will eventually be problematic), we studied in a second time the data storage of the main client of Ethereum (Geth) and its way of synchronizing the state of the blockchain between the peers. We have designed a new distributed storage architecture for Ethereum taking advantage of the DHT, backward compatible with the current clients and able to reduce the disk space used for long-term storage by 95% (58% of the total storage), without impacting the guarantees or the performance of the Ethereum blockchain.

However, storing data on the DHT makes it more prone to attacks, especially Sybil attacks. We therefore analyzed Ethereum peers for patterns that could reflect Sybil attacks and showed the existence of thousands of suspicious nodes grouping many identifiers for a single IP address (up to 10000/IP). We finally designed and implemented a protection architecture against Sybil attacks. It is based on a crawler detecting suspicious nodes in real time, a smart contract structuring the information and distributing it to all peers, and finally a fully-distributed revocation system, each peer noticing itself the attack and removing its connections. The deployment on an Ethereum test network has shown the effectiveness of the proposed architecture.

Keywords: blockchain, Bitcoin, Ethereum, P2P network, reliability, monitoring, Sybil attack, attack detection, revocation system, distributed storage, distributed hash table

Résumé

Analyse, valorisation et protection des réseaux pair-à-pair de blockchains publiques

Les blockchains reposent sur des réseaux P2P essentiels à leur bon fonctionnement puisqu'ils assurent la dissémination des transactions et des blocs à l'ensemble des parties. Alors que Bitcoin et Ethereum – les deux principales blockchains publiques – capitalisent aujourd'hui des milliers de milliards de dollars, attirant chaque jour de nouveaux utilisateurs, peu d'études s'intéressent aux aspects réseau bien que la littérature montre que de nombreux problèmes peuvent réduire la fiabilité des réseaux P2P publics.

Dans cette thèse, nous nous sommes intéressés dans un premier temps à la supervision des réseaux P2P des blockchains Bitcoin et Ethereum. Nous avons implanté un crawler pour chaque réseau capable de découvrir tous les pairs connectés et avons analysé les données issues de campagnes de mesure de plusieurs mois. Différents critères pouvant affecter la fiabilité du réseau ont été étudiés, tels que le nombre de pairs, leur distribution du point de vue géographique ou du réseau IP, leur taux d'attrition, la proportion de clients aux vulnérabilités connues, l'existence de motifs journaliers de connexion ou encore la capacité d'inférer la topologie. Il apparaît que les deux réseaux montrent de bonnes propriétés sur tous ces points.

Partant du constat que, d'une part, le réseau P2P d'Ethereum basé sur une table de hachage distribuée (DHT) est largement inexploité, car aucune donnée n'est stockée dans la DHT, et que, d'autre part, le stockage des données de la blockchain ne fait que croître (ce qui posera des problèmes à terme), nous avons étudié dans un second temps le stockage des données du principal client d'Ethereum (Geth) et sa manière de synchroniser l'état de la blockchain entre les pairs. Nous avons conçu une nouvelle architecture distribuée de stockage pour Ethereum tirant parti de la DHT, rétrocompatible avec les clients actuels et pouvant réduire l'espace disque, utilisé pour le stockage long terme, de 95% (58% du stockage total) sans impact sur les garanties ou les performances de la blockchain Ethereum.

Le stockage des données sur la DHT la rend cependant plus intéressante à attaquer, en particulier par des attaques Sybil localisées. Nous avons donc analysé les pairs d'Ethereum à la recherche de motifs pouvant traduire des attaques Sybil et montré l'existence de milliers de nœuds suspects regroupant un grand nombre d'identifiants pour une même adresse IP (jusqu'à 10000/IP). Nous avons finalement conçu et implanté une architecture de protection contre les attaques Sybils. Celle-ci se base sur un crawler détectant les nœuds suspects en temps réel, un smart contract structurant l'information et la distribuant à tous les pairs, et enfin une révocation complètement distribuée, chaque pair constatant lui-même l'attaque et coupant ses connexions aux nœuds Sybils. La mise en œuvre sur un réseau de test Ethereum a montré l'efficacité de l'architecture proposée.

Mots-clés: blockchain, Bitcoin, Ethereum, réseau P2P, fiabilité, supervision, attaque Sybil, détection d'attaque, mécanisme de révocation, stockage réparti, table de hachage distribuée