

Projet Pluridisciplinaire d'Informatique Intégrative

InterPlanetary File System

Loris Alexandre
Sangoan Brigué
Hugo Pagniez

Année 2021–2022

Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Alexandre, Loris

Élève-ingénieur(e) régulièrement inscrit(e) en 1^e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31903210

Année universitaire : 2021–2022

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

InterPlanetary File System

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

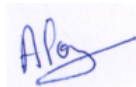
Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 7 mars 2022

Signature :



Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Brigué, Sangoan

Élève-ingénieur(e) régulièrement inscrit(e) en 1^e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 31902075

Année universitaire : 2021–2022

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

InterPlanetary File System

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 7 mars 2022

Signature :



Déclaration sur l'honneur de non-plagiat

Je soussigné(e),

Nom, prénom : Pagniez, Hugo

Élève-ingénieur(e) régulièrement inscrit(e) en 1^e année à TELECOM Nancy

Numéro de carte de l'étudiant(e) : 110902578

Année universitaire : 2021–2022

Auteur(e) du document, mémoire, rapport ou code informatique intitulé :

InterPlanetary File System

Par la présente, je déclare m'être informé(e) sur les différentes formes de plagiat existantes et sur les techniques et normes de citation et référence.

Je déclare en outre que le travail rendu est un travail original, issu de ma réflexion personnelle, et qu'il a été rédigé entièrement par mes soins. J'affirme n'avoir ni contrefait, ni falsifié, ni copié tout ou partie de l'œuvre d'autrui, en particulier texte ou code informatique, dans le but de me l'accaparer.

Je certifie donc que toutes formulations, idées, recherches, raisonnements, analyses, programmes, schémas ou autre créations, figurant dans le document et empruntés à un tiers, sont clairement signalés comme tels, selon les usages en vigueur.

Je suis conscient(e) que le fait de ne pas citer une source ou de ne pas la citer clairement et complètement est constitutif de plagiat, que le plagiat est considéré comme une faute grave au sein de l'Université, et qu'en cas de manquement aux règles en la matière, j'encourrais des poursuites non seulement devant la commission de discipline de l'établissement mais également devant les tribunaux de la République Française.

Fait à Nancy, le 7 mars 2022

Signature :



Table des matières

Table des matières	iv
1 Introduction	1
2 Adressage des données par le contenu	2
2.1 Adressage des données	2
2.2 Hachage	2
2.3 Immuabilité des données	3
2.4 Persistence des données et recyclage de la mémoire	3
3 Merkle DAG	4
4 P2P : Table de hachage distribuée et routage	5
5 IPFS en action	7
5.1 Ajout de fichier	7
5.2 Utilisation avancée	7
5.3 Commande P2P	8
6 Conclusion	9
Bibliographie / Webographie	10
Liste des illustrations	11
Acronymes	12

1 Introduction

IPFS est un système distribué de fichiers pair à pair conçu par Juan Benet. Le principe est simple : on associe une adresse statique (un identifiant) au fichier que l'on souhaite héberger sur le système, et on peut grâce à celui-ci y accéder quand l'on souhaite, peu importe où il est stocké. Pour créer l'identifiant, on se base sur le contenu de la donnée et on la passe à travers une fonction de hachage, ce qui permet d'avoir des identifiants uniques. Ce système permet alors de mettre en place un mécanisme de décentralisation, ce qui permet d'éviter les point unique de défaillance (ou "Single Point Of Failure" en anglais) car les données peuvent être hébergées à plusieurs endroits à la fois.

En d'autres termes, lorsqu'un utilisateur souhaitera récupérer un fichier, il va d'une part se connecter automatiquement à travers **IPFS** à des personnes ayant déjà le fichier en question, et une fois récupérée, il sera lui aussi dans la liste de personnes ayant ce fichier et donc en devient un hébergeur.

Nous allons à travers ce rapport présenter les concepts et mécanismes clés de l'**IPFS**, qui sont : l'identification du contenu, l'utilisation de graphe (**DAG**), et l'accès au contenu via le pair-à-pair (**P2P**). Nous terminerons par une petite démonstration de l'**IPFS** afin de comprendre son principe d'utilisation.

2 Adressage des données par le contenu

2.1 Adressage des données

Chaque élément enregistré dans l'**IPFS** est relié à un **CID**. Un **CID**, pour "Content Identifier" ou "Identifiant de Contenu" en français, ne fournit pas l'information d'où est-ce qu'est sauvegardé l'élément auquel il est rattaché, mais permet néanmoins de le retrouver car il est construit à partir du contenu de cet élément. Bien sûr, la taille de cet identifiant est minime par rapport au contenu de l'élément qu'il définit. Le fait qu'il soit basé sur le contenu d'un élément implique que n'importe quel changement sur ce contenu engendrera un changement de **CID**. De la même façon, un même fichier sauvegardé sur deux différents noeuds **IPFS** aura le même **CID**.

On distingue deux versions de format de cet identifiant. La première, nommée CIDv0, est un multihash encodé en base 58 et est toujours utilisé pour un grand nombre d'opérations **IPFS** basiques. Autrement dit, si un **CID** est composé de 46 caractères et qu'il commence par "Qm", c'est qu'il s'agit d'un CIDv0. La seconde version, nommée CIDv1, est composée de multiples identifiants préfixant le contenu haché qui permettent d'apporter un grand nombre d'informations sur celui-ci. Premièrement, il y a un préfixe multibase qui spécifie le type d'encodage utilisé pour encoder le reste du **CID**, ensuite un identifiant de version du **CID** et enfin un identifiant multicodec qui indique le format de l'élément qu'identifie le **CID**. Cette version va bientôt être considérée comme par défaut par le projet **IPFS**.

2.2 Hachage

En cryptographie, les fonctions de hachage permettent de prendre n'importe quel type de donnée en entrée afin d'en retourner une valeur à taille fixe. Cette valeur de retour dépend bien évidemment de l'algorithme de hachage utilisé. Ces haches peuvent être encodées en différentes bases. Comme vu précédemment, dans la version 1 du **CID**, **IPFS** supporte ces différents types d'encodage à travers la définition du préfixe multibase.

Les haches sont définis par plusieurs caractéristiques qu'il ne faut pas négliger. En effet, le résultat du hachage d'un même message par la même fonction de hachage sera toujours identique (déterminisation); n'importe quelle modification d'un message, même minime, produira un hache complètement différent (indépendance); il est impossible de générer un même hache avec deux messages différents (unicité); une fois un message haché, il n'est plus possible de retrouver le message initial (sens unique). Ces caractéristiques signifient qu'un hache peut être utilisé pour identifier n'importe quelle sorte de donnée puisqu'il est unique et de taille raisonnablement courte, ce qui permet de le faire naviguer à travers le réseau sans conséquences de performances. Cette dernière remarque est critique au bon fonctionnement de l'**IPFS** car il est essentiel de rechercher

rapidement un fichier à travers le réseau et il ne serait pas concevable de faire sans un identifiant unique à chacun de ces fichiers.

2.3 Immuabilité des données

Un objet immuable est un élément dont l'état ne peut être altéré ou modifié une fois qu'il est créé. A travers **IPFS**, cela signifie, qu'une fois qu'un fichier est ajouté au réseau, il n'est pas possible de le modifier sans également modifier son **CID**. On rencontre alors un sérieux problème lorsqu'il s'agit de satisfaire ce besoin. Le **CID** est donc immuable peu importe le contexte dans lequel il est utilisé. On considère ce **CID** comme un pointeur vers un élément du réseau. Pour pouvoir utiliser des objets immuables dans un environnement muable, il faut ajouter un autre principe en surcouche. On peut par exemple citer le système de l'**IPNS**, pour "InterPlanetary Name System" qui permet d'associer à une adresse fixe un **CID** qui peut être modifié à travers le temps, ce qui permet de modifier une donnée et d'y accéder via la même adresse que la version précédente.

2.4 Persistance des données et recyclage de la mémoire

L'un des objectifs de l'**IPFS** est de préserver l'histoire de l'humanité en permettant aux utilisateurs de sauvegarder leurs données en minimisant la possibilité qu'elles disparaissent accidentellement. **IPFS** permet en réalité de conserver chacune des versions du fichier sauvegardé et de le retrouver facilement.

Lorsqu'un fichier est téléchargé, il est automatiquement enregistré dans le cache pour permettre d'y accéder plus rapidement la prochaine fois. Ce système dépend en réalité du nombre de nœuds qui souhaite participer au cache et au partage des données. Si on part sur ce principe, le stockage est infini donc les nœuds ont besoin de faire de la place pour mettre en cache les nouvelles ressources fraîchement téléchargées en supprimant les plus anciennes ou les moins populaires. Ce principe est appelé 'garbage collection'.

Le principe de 'garbage collection' est une forme de gestion automatique des ressources largement utilisée dans n'importe quelle solution de développement logicielle. La 'garbage collection' tente de recycler la mémoire utilisée par la donnée qui n'est plus exploitée. Au sein de l'**IPFS** ce principe est utilisé pour libérer de l'espace mémoire d'un nœud en supprimant les données qui ne nécessitent plus d'être conservées.

Pour garantir qu'un fichier persiste dans l'**IPFS**, il est nécessaire d'utiliser le principe d'accrochage. L'accrochage permet de gérer manuellement l'espace mémoire d'un nœud pour retenir de la donnée et éviter que la garbage collection ne la supprime/remplace. Un cas d'usage majeur de cette fonctionnalité est la conservation indéfini d'un fichier.

3 Merkle DAG

On a vu précédemment que pour chaque fichier que l'on ajoute au système **IPFS**, une adresse d'identification lui ai donné. Mais en réalité, il y a une autre étape qui est effectuée entre le moment d'import et le moment où on nous donne l'identifiant : le découpage du fichier en de multiples sous-parties.

En réalité, un **CID** ne référence pas un fichier, mais un nœud dans un graphe orienté acyclique que l'on appelle **DAG** (Directed acyclic graph). Chaque nœud possède alors :

- Une adresse d'identification (**CID**)
- Des données (**le payload**)
- Une liste de sous-nœuds (**les fils du nœud courant**)

Regardons de plus près le Merkle DAG suivant :



FIGURE 3.1 – Exemple de Merkle DAG

On voit donc sur l'illustration que nous avons voulu ajouter un fichier de 506 kb à notre système **IPFS**, comme il dépasse la taille maximale par défaut de l'**IPFS**, c'est-à-dire 256 kb, il a alors été découpée en deux nœuds (0 et 1). Ces deux nœuds contiennent alors une quantité de données de respectivement 256 kb et 250 kb (en additionnant les quantités, on retrouve bien la taille du fichier initial). En haut de l'arbre, nous avons le nœud parent qui contient le **CID** qui est l'unique référence du fichier complet, et on retrouve également les différents liens vers ses sous-nœuds. Ce principe permet alors de propager des données plus rapidement, en effet, une personne n'est plus obligée d'héberger l'intégralité du fichier pour le rendre accessible.

Ce principe permet alors de propager des données plus rapidement, en effet, une personne n'est plus obligée d'héberger l'intégralité du fichier pour le rendre accessible. Dans le cas présent, on peut par exemple imaginer qu'une personne stocke le nœud 1 sur son ordinateur, et que le nœud 0 soit stocké chez un autre utilisateur. Lors de la récupération du fichier par un tiers, **IPFS** récupérera les différents nœuds liés à un **CID**, et cela où qu'ils soient stockés.

4 P2P : Table de hachage distribuée et routage

Comme vu précédemment, on sait que les différentes données sont hébergées par petit bout et que chaque personne peut en avoir une partie. Pour trouver quel pair héberge le contenu auquel on souhaite accéder, on utilise un système de table de hachage. Une table de hachage est une base de données dans laquelle chaque clé est associée à une valeur. Lorsque l'on souhaite récupérer le contenu lié à un certain **CID**, on cherche à savoir quels pairs l'héberge, on récupère alors cette information via cette table de hachage. Une fois que les pairs sont identifiés, on utilise une autre table de hachage pour trouver leurs locations et leurs distances : c'est la table de routage.

Cependant, **IPFS** utilise une autre version : les tables de hachage distribuées (**DHT**). Elle permet d'être utilisée à travers toutes les pairs du réseau. Pour construire cette table de hachage distribuée et notamment gérer la table de routage, **IPFS** utilise l'algorithme Kademlia.

Dans Kademlia, chaque pair du réseau est identifié de façon unique (dans le cas d'**IPFS**, on utilise tous les nombres de 0 à $2^{256}-1$). Chaque pair dispose alors d'une adresse pour pouvoir être contacté ainsi que de sa propre table de routage contenant les adresses des pairs qu'il connaît. Au démarrage, un pair va automatiquement se connecter à des pairs prédéfinis afin de leur demander des informations sur les autres pairs qu'ils connaissent.

Dans la table de routage, on retrouve des informations sur les autres pairs réseaux. Cependant, on ne peut pas stocker les informations sur toutes les pairs du réseau. En effet, les pairs viennent et partent du réseau, il serait donc difficile de maintenir à jour la table de routage. De plus, la taille de pairs stockées est limitée au nombre de bits qui composent l'identifiant unique. La table de routage ne stocke donc que les pairs qui sont à 2^N de distance, N étant le nombre maximum de pairs à enregistrer. On peut représenter les pairs du réseau via un arbre binaire tel qu'il suit :

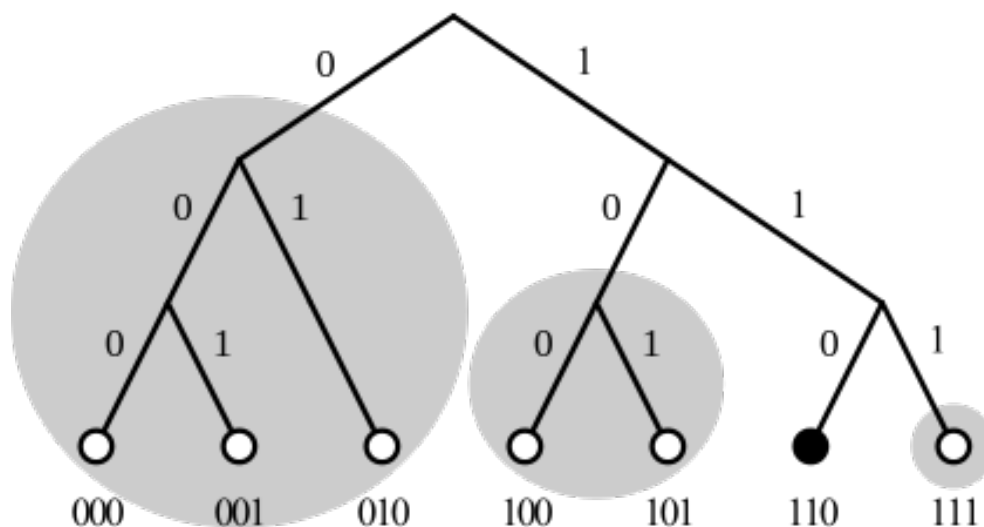


FIGURE 4.1 – Arbre binaire Kademlia

Ici, on peut représenter des sous-groupes de pairs. Chaque pair doit référencer dans sa table de routage au moins une pair de chaque sous-groupe. Cela permet de retrouver chaque pair lors de l'algorithme de recherche. Cependant, la distance reste une notion abstraite.

Kademlia définit alors sa propre fonction de distance, celle-ci n'a aucun rapport avec la distance physique entre deux pairs. La fonction distance utilisée est l'opérateur XOR, si on cherche par exemple à déterminer la distance entre le pair 10 et le pair 13, on obtient une distance de 7. En effet, $10 \text{ XOR } 13 = 7$.

On peut détailler le calcul en passant en binaire : $001\ 010 \text{ XOR } 001\ 101 = 000\ 111$

Ainsi, si on prend un réseau de 16 pairs et que le pair 0 souhaite faire un appel au pair 13, on va calculer la distance entre les pairs présents dans la table de routage et le pair 13. On a donc $1 \text{ XOR } 13 = 12$, $2 \text{ XOR } 13 = 15$, $4 \text{ XOR } 13 = 9$, $8 \text{ XOR } 13 = 5$. On effectue alors un appel au pair 8 pour lui demander de trouver le pair 13. Il va réaliser la même opération avec les pairs de sa table de routage, et on effectue ce processus jusqu'à trouver le chemin vers le pair 13.

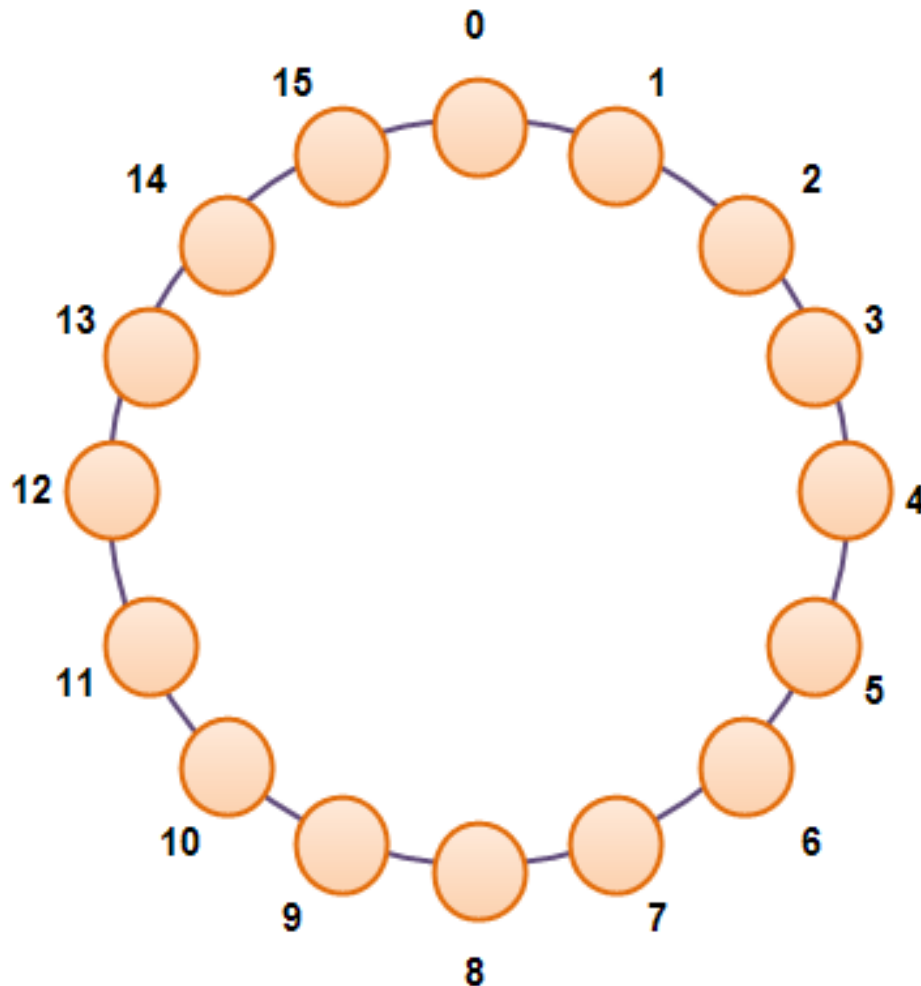


FIGURE 4.2 – Réseau de 16 pairs

Dans le cas où un pair viendrait à quitter le réseau, on va chercher à le pair le plus proche du pair sortant pour le remplacer. Grâce à cet algorithme, on peut retrouver rapidement un contenu. D'un point de vue mathématique, la complexité de la recherche est estimée à $O(n \log(n))$.

5 IPFS en action

Pour cette partie, nous allons partir de la version console d'**IPFS** et non pas de l'interface graphique qu'il nous mette à disposition (IPFS Desktop). Une fois la **CLI** installé, nous pouvons initialiser le système **IPFS** avec la commande suivante :

```
1 ipfs init
```

Le but de cette commande est de nous générer un identifiant de pair qui permettra aux autres utilisateurs de nous contacter, comme vu dans la partie précédente.

5.1 Ajout de fichier

Maintenant que nous pouvons utiliser **IPFS**, nous allons ajouter un fichier **LICENSE** présent dans le dossier courant au réseau, on utilise pour cela la commande :

```
1 ipfs add LICENSE
```

En effectuant cette commande, **IPFS** va nous retourner l'adresse associé au fichier que nous venons d'ajouter. Nous pouvons alors utiliser cette référence pour récupérer le contenu du fichier, via la commande :

```
1 ipfs cat <CID>
```

ou encore consulter les liens vers les différents noeuds qui le compose (principe des **DAGs**) avec la commande :

```
1 ipfs ls <CID>
```

5.2 Utilisation avancée

Des commandes plus poussées sont également présentes : nous pouvons en effet consulter des informations supplémentaires pour chacun des éléments que nous ajoutons au système **IPFS**. Nous pouvons consulter le **DAG** associé à chacun des **CID**, ainsi que consulter les différentes informations sur chacun des blocs contenue dans un **DAG**.

```

1 // Liste des adresses des sous-noeuds composant un élément
2 ipfs refs <CID>
3
4 // Informations sur le DAG d'un élément :
5 ipfs dag get <CID>
6
7 // Informations sur un bloc contenu dans un DAG :
8 ipfs block get <CID>

```

5.3 Commande P2P

Il y a également plusieurs commandes qui concerne la partie P2P d'**IPFS**, en voici quelques exemples :

```

1 // Commande concernant la connexion avec les pairs :
2 ipfs swarm <>
3 - addrs : consulte les adresses déjà connues
4 - connect <adresse> : permet de se connecter à un pair
5 - disconnect <adresse> : permet de se déconnecter d'un pair
6 - peers : permet de voir les pairs avec une connexion active
7
8 // Commande concernant la table de hachage distribuée :
9 ipfs dht <>
10 - findpeer <id> : récupération d'une adresse multicast via un identifiant
11 - get <CID> : permet de connaître le routage pour récupérer un élément
12 - provide <CID> : permet d'informer le réseau que nous avons un certain CID

```

6 Conclusion

Nous avons étudié en détail les trois concepts clés de l'InterPlanetary FileSystem. Tout d'abord, chaque contenu dans **IPFS** est identifié de façon unique grâce à **CID**. Ensuite, il est découpé en plusieurs parties dans un Merkle **DAG**. Enfin, on retrouve ce contenu à travers le réseau grâce à l'algorithme Kademlia et le système de **DHT**. Nous avons aussi étudié les principales commandes d'**IPFS** pour comprendre au mieux son fonctionnement.

Le but est désormais d'implémenter un système de fichier décentralisée, version TelecomNancy. Nous utiliserons nos connaissances en réseau, en base de données et en C pour y parvenir. Cette étude nous aura permis de comprendre le fonctionnement d'**IPFS** et de définir des objectifs d'implémentation.

Bibliographie / Webographie

- [1] Peter Maymounkov et David Mazières. Kademlia : A peer-to-peer information system. <https://pdos.csail.mit.edu/~petar/papers/maymounkov-kademlia-lncs.pdf>, 2022. Dernier accès le 06 Mars 2022.
- [2] IPFS. Ipfs documentation. <https://docs.ipfs.io/>, 2022. Dernier accès le 06 Mars 2022.
- [3] Jakob Jenkov. P2p network algorithms. <https://jenkov.com/tutorials/p2p/index.html#p2p-network-algorithms>, 2022. Dernier accès le 06 Mars 2022.
- [4] Eleuth P2P. Kademlia algorithm presentation. <https://youtu.be/7o0pfKDq9KE>, 2022. Dernier accès le 06 Mars 2022.

Liste des illustrations

3.1	Exemple de Merkle DAG	4
4.1	Arbre binaire Kademia	5
4.2	Réseau de 16 pairs	6

Acronymes

CID Content Identifier

CLI Command Line Interface

DAG Directed Acyclic Graph

DHT Distributed Hash Table

IPFS InterPlanetary File System

IPNS InterPlanetary Name System

P2P Pair-à-pair (peer-to-peer)