POLYTECH SORBONNE

SCIENCES SORBONNE UNIVERSITÉ

Rapport projet d'initiation : Blockchains (IOTA)

UE : Projet d'initiation MAIN

Étudiants :
Tahar AMAIRI
Léo LASSALLE
Émile PICHARD
Ghassen HACHANI

Encadrants :
Maria POTOP-BUTUCARU
Gewu BU

Responsable UE :
Cécile BRAUNSTEIN

Groupe :
MPB-2

17 Mai 2021

Table des matières

1 Introduction

Dans le cadre de notre projet d'initiation, nous avons travaillé sur le sujet des crypto-monnaies et plus précisément sur la crypto-monnaie **IOTA**. De son implémentation théorique à des tentatives d'amélioration de son fonctionnement, nous nous sommes basés sur le code déjà existant du répertoire git [5].

Une crypto-monnaie est une monnaie virtuelle qui s'échange de pair en pair sans l'intervention de banques ou d'autres acteurs intermédiaires selon un protocole spécifique dont le plus populaire est appelé **blockchain**. Une blockchain est une technologie de stockage et de transmission d'informations. En soit, il s'agit d'une base de données distribuée dont la gestion est traitée par un réseau d'ordinateurs inter-connectés, où chaque transaction va être d'abord être validée par un utilisateur du système avant d'être enregistrée dans la chaîne contre une commission.

La technologie blockchain permet d'assurer la fiabilité et la traçabilité des transactions. Les crypto-monnaies sont apparues en 2009 avec le Bitcoin, créé par un programmeur sous le pseudonyme de Satoshi Nakamoto. Cependant, aujourd'hui de nombreuses crypto-monnaies sont apparues sur le marché ayant chacune un protocole unique. Mais beaucoup d'entre elles, et notamment les plus populaires, utilisent des protocoles blockchains très similaires.

Ces crypto-monnaies présentent 3 avantages majeurs par rapport à des monnaies traditionnelles. Premièrement, elles sont très sécurisées grâce à la blockchain qui est un système presque inviolable puisqu'il faudrait pirater de nombreuses bases de données différentes, chacune doublement sécurisées. Elles sont aussi transparentes, c'est à dire que puisque les membres du réseau valident et vérifient les transactions, les fraudes peuvent être aisément repérées. Dernièrement, les crypto-monnaies sont autonomes puisque les utilisateurs se vérifient entre eux et donc suppriment l'intermédiaire humain qui peut être source de méfiance mais aussi de perte d'argent.

Néanmoins, ces crypto-monnaies ont aussi des défauts. Pour commencer, la blockchain est très inégalitaire puisqu'elle se base sur la résolution de problèmes cryptographiques et privilégie ainsi les utilisateurs ayant le plus de puissance de calcul d'où une consommation énergétique élevée et des frais pouvant excéder le montant d'une transaction.

Ainsi, ces désavantages peuvent freiner la démocratisation de ces crypto-monnaies dans certains domaines tels que **l'IoT** (Internet Of Things). C'est donc dans l'optique d'effacer ces défauts que des crypto-monnaies comme IOTA ont émergé.

Nous étudierons à travers ce papier le fonctionnement d'IOTA et nous discuterons par la suite de différents algorithmes de sélection de transactions et de leur efficacité.

2 IOTA

IOTA est une crypto-monnaie introduite en 2017 dont le but est de pouvoir effectuer des transactions entre utilisateurs. Et comme toute crypto-monnaie, IOTA a besoin d'un système de stockage et de validation des transactions. Comme dit précédemment, la grande majorité des crypto-monnaies actuelles utilisent le protocole blockchain.

L'intérêt majeur d'IOTA est de ne pas utiliser cette architecture blockchain afin d'éviter les désavantages inhérents à celle-ci et d'utiliser plutôt une architecture moins conventionnelle appelée **Tangle**.

2.1 La structure du Tangle

Le Tangle prend la forme d'un **DAG** (Directed Acyclic Graph) qui se construit de gauche à droite au fil du temps ayant la spécificité que chaque entité le constituant est à la fois "validateur" et "utilisateur" du système. Il est important de noter que le fonctionnement du réseau d'IOTA est **asynchrone** et **décentralisé**, i.e, que chaque nœud du réseau possède son propre Tangle local et lors du changement de celui-ci, le nœud responsable diffuse son Tangle local aux autres nœuds du réseau pour qu'ils mettent à jour le leur.

Voici un exemple simplifié du Tangle :

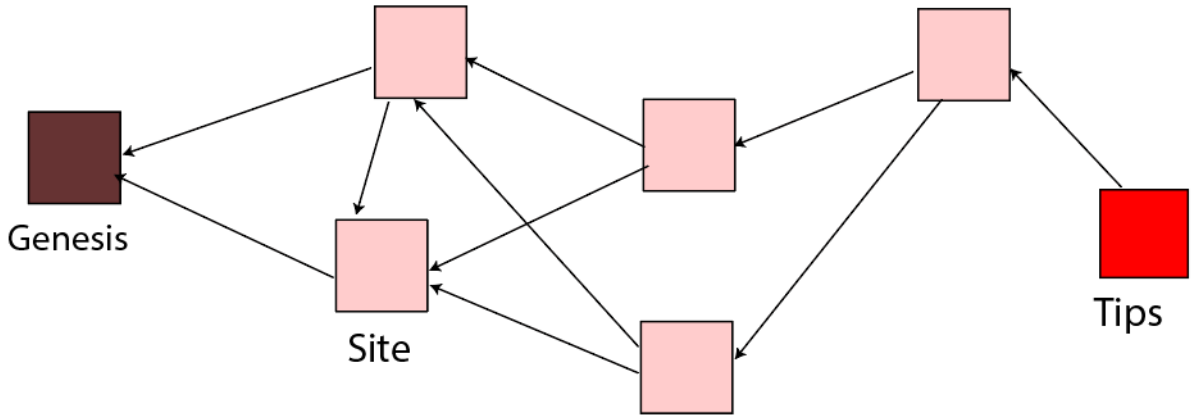

FIGURE 1 – Schéma d'un Tangle simplifié

Légende :

- `Genèse` : c'est l'adresse fondatrice du Tangle et la première transaction a être émise dans le Tangle (elle est donc approuvée par toutes les autres transactions).
- `Site` : une ancienne transaction sauvegardée dans le Tangle.
- `Tip` : une transaction non encore approuvée émise par un nœud dans le Tangle.

Une petite note sur la terminologie : les sites sont les transactions représentées dans le Tangle (elles sont donc en quelque sorte les nœuds d'un graphe). A contrario, le réseau est composé de nœuds (c'est à dire d'utilisateurs), ce sont les entités qui émettent et valident les transactions. Ainsi, un site contient les informations d'une transaction, i.e, son montant, l'identité de son émetteur, de son receveur, la date...

Bien plus, chaque transaction possède ce qu'on l'on appelle un **poids** :

- `Poids propre` : chaque transaction possède un poids propre qui est proportionnel à la quantité de travail effectuée par le nœud émetteur pour émettre celle-ci.
- `Poids cumulé` : chaque transaction possède également un poids cumulé valant son poids propre et celui de toutes les transactions qui l'approuvent directement et indirectement.

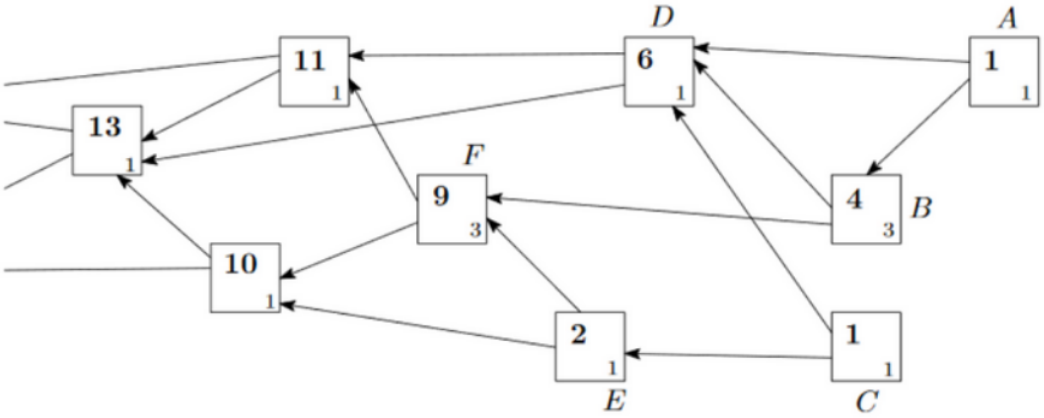

FIGURE 2 – Poids propre et cumulé

Dans cet exemple, le site F possède un poids propre égal à 3 et un poids cumulé égal à son poids propre plus le poids propre des sites E, C, B et A (car ces sites approuvent de façon directe ou indirecte F), ce qui vaut 9.

"**Approuver**" une transaction se reflète sur le Tangle par une flèche reliant deux sites : par exemple, F est approuvé *directement* par B car il existe une flèche partant de B vers F. Bien plus, C approuve aussi F mais *indirectement* car il existe un chemin allant de C à F (en passant par le site E).

Les transactions étant validées par les utilisateurs, on ne peut pas être convaincu de la validité d'une transaction. Cependant, une transaction ayant un poids cumulé important a de très grandes chances d'être "légitime". Nous verrons par la suite comment IOTA empêche ce type de comportement malhonnête.

Pour émettre une transaction dans IOTA, un utilisateur doit :

1. Envoyer la transaction depuis son appareil.
2. Créer un tip (correspondant à cette dite transaction) dans le Tangle en choisissant 2 transactions à approuver de façon aléatoire (en utilisant un algorithme).
3. L'appareil émetteur vérifie que ces transactions sont cohérentes avec son Tangle local.
4. Finalement, l'émetteur valide les deux transactions choisies en résolvant une énigme cryptographique (*Proof of Work* i.e PoW).

Ainsi, pour qu'une transaction soit validée, elle doit approuver 2 tips du Tangle. S'il y a discordance entre l'historique local de deux nœuds, le système statue sur quelle transaction garder et quelle transaction écarter du Tangle. Ainsi, la sécurité du système est théoriquement assurée.

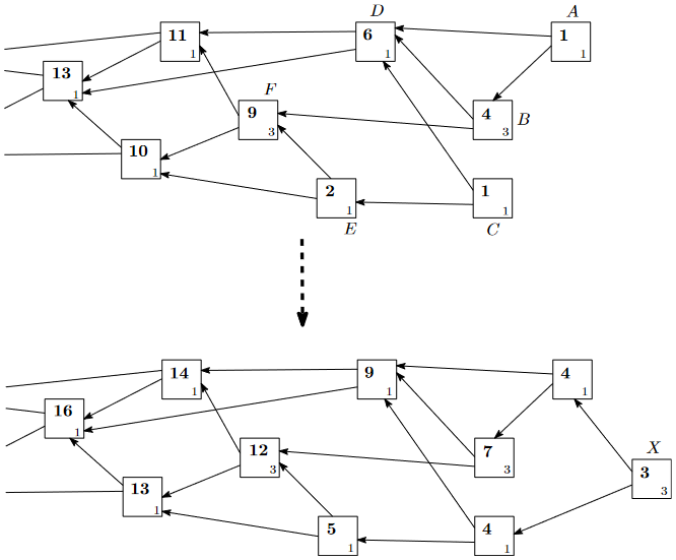

FIGURE 3 – Ajout d'une nouvelle transaction au Tangle

Dans la figure 3, les sites A, B et C sont des tips avant l'émission de la transaction X. Lors de l'ajout de celle-ci au Tangle, elle approuve à son tour les transactions A et C. Finalement, les nouveaux tips sont le site X et B car celui-ci n'a pas encore été approuvé. Notons aussi le changement des poids propres et cumulés de chacun des sites suite à l'ajout de la transaction X.

Par conséquent, pour pouvoir émettre des transactions, un utilisateur est dans l'obligation de participer à la sécurité du Tangle, sinon il ne verra jamais ses transactions validées par les autres utilisateurs.

2.2 La sélection des tips

L'enjeu principal est donc de savoir choisir les tips à approuver de manière aléatoire tout en garantissant la sécurité du Tangle face à de potentielles attaques (discutées dans le papier [6]), en ayant un taux de transactions validées satisfaisant. Pour cela, on introduit les **Tips Selection Algorithm** (TSA) utilisant une **Markov Chain Monte Carlo** (MCMC). Ainsi, on implémentera les versions de la marche aléatoire [1] des papiers [6], [2] et [3].

1. Un GIF d'une marche aléatoire dans un Tangle

3 Simulation du Tangle

Pour simuler la structure du Tangle afin d'implémenter les différents TSA, nous allons utiliser le git [5]. Celui-ci se base sur un logiciel nommé **OMNeT++** qui est une bibliothèque et un cadre de simulation réseau écrit en **C++**.

Tout projet utilisant OMNeT++ doit contenir trois types de fichiers :

— Un fichier **NED** permettant de définir les composantes du réseau et les paramètres de la simulation. Dans notre cas c'est le fichier `TangleSim.ned`, définissant deux modules : **TxActorModule** représentant les nœuds du réseau d'IOTA et **TangleModule** représentant la structure du Tangle. Il est important de noter que la simulation est centralisée autour de ce dernier, ce qui est en désaccord avec le fonctionnement réel d'IOTA cependant cela n'impactera pas nos résultats car nous étudions dans ce papier l'efficacité des TSA.

— Des fichiers **C++** qui contiendront le code à exécuter par les modules. Pour notre simulation, nous avons le header `Tangle.h` qui contient les déclarations de structures et de classes (des nœuds et des sites), `TangleModules.cc` qui contient le code exécuté par le module TangleModule et `Tangle.cc` qui contient le code exécuté par le module TxActorModule.

— Et finalement un fichier **INI** permettant d'initialiser la simulation et sa configuration (nombre de fois qu'on veut simuler, le temps d'exécution, l'interface de simulation (Tkenv (IDE), Qtenv (IDE) ou bien Cmdenv (console), fichier de log etc...). Dans notre cas c'est le fichier `omnet.ini`.

Le git [5] de référence permet donc de simuler le Tangle, mais n'implémente aucun suivi détaillé de la marche aléatoire ou bien de la construction de celui-ci. Nous présenterons donc, dans la prochaine section, les moyens mis en place pour suivre la marche aléatoire et la visualisation du Tangle.

Toutes les modifications apportées à ce git de référence et les implémentations qui suivront se trouveront sur le git de notre projet [1].

4 Visualisation

OMNeT++ possède des fonctions d'affichages avancées dans son IDE/GUI. Cependant, elles sont complexes à utiliser et nous allons donc privilégier des fichiers **logs au format .txt**. Ces fichiers sont créés lorsque par exemple une marche aléatoire est initiée ou bien lorsque que le Tangle change d'un temps t à un temps $t + 1$.

Par exemple :

```
* TSA Path:
   - Site root: Genesis
       > Current site: Genesis, weight: 4, walk counts: 0
       > Adjacent site(s):
           -> Site: 1, weight: 2, probability: 0.689974
           -> Site: 2, weight: 1, probability: 0.310026
       > Uniform probability: 0.698195, selected site for the next walk => 1

       > Current site: 1, weight: 2, walk counts: 1
       > Adjacent site(s):
           -> Only one adjacent site available after the call of filterView():
              Site: 3

       > Current site: 3, weight: 1, walk counts: 2
       > Adjacent site(s):
           -> No adjacent sites available after the call of filterView()

   - Selected tip for potential approval: 3, total walk count: 3
```

FIGURE 4 – Fichier log TSATrackeractors[1].txt

Le fichier de la figure 4 permet de suivre l'état de la marche aléatoire initiée par le nœud n°1, on y remarque : le site d'origine sur lequel le TSA a été initié, la probabilité de passage d'un site à un autre, le nombre de marche effectué, le poids des sites, les sites adjacents et finalement le tip sélectioné.

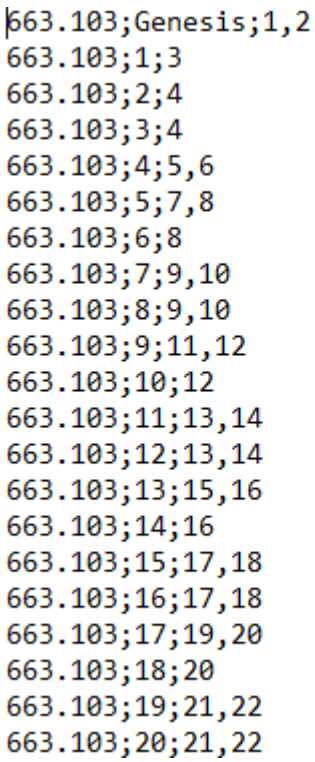

FIGURE 5 – Fichier log TrackerTangle.txt

Le fichier de la figure 5 permet quant à lui de connaître la composition du Tangle en un temps donné en utilisant ce format **CSV** : temps de la simulation ;site père ;site(s) fils approuvant le père directement. Par exemple pour la première ligne, le temps de la simulation est de 663.103 secondes, le site père est le Genesis et celui-ci est approuvé par les sites 1 et 2.

Ce fichier permet aussi de visualiser la forme du Tangle graphiquement. Pour cela, nous utilisons le langage **DOT**, un langage de description de graphe.

Pour convertir le fichier log TrackerTangle.txt en un code DOT, nous allons utiliser **Python** et le module **Graphviz**. Ainsi, le script python `TangleGen.py` retranscrit les fichiers .txt créés par la simulation OMNeT++ en .dot. Finalement, le rendu est obtenu avec le moteur Graphviz.

Voici un exemple du rendu :

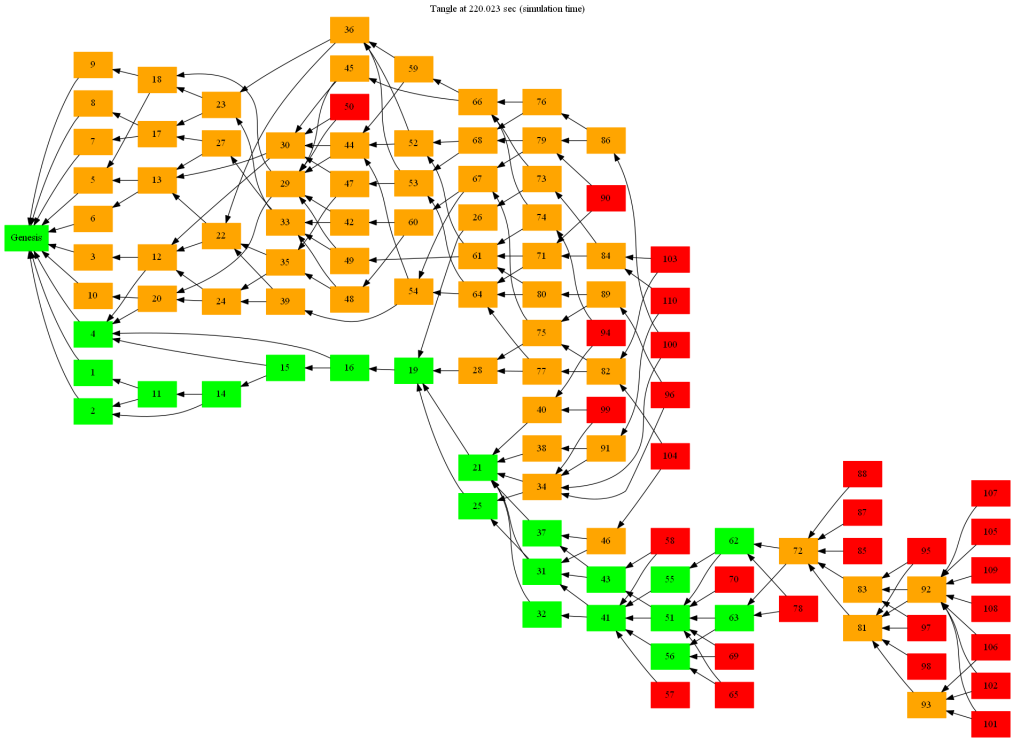

FIGURE 6 – Affichage d'un Tangle via TangleGen.py

Légende :

— : une transaction validée ayant une probabilité **supérieure** à 50% d'être approuvée indirectement par l'ensemble des tips.

— : une transaction validée ayant une probabilité **inférieure** à 50% d'être approuvée indirectement par l'ensemble des tips.

— : un tip.

L'intérêt d'implémenter tous ces fichiers logs est de pouvoir vérifier le bon fonctionnement des TSA et d'obtenir un moyen de visualiser le Tangle.

5 Critères d'évaluation

Afin de comparer l'efficacité des algorithmes que nous implémenterons, nous sélectionnerons trois critères d'évaluations :

— **Le temps d'exécution de la simulation** : ce critère permet d'évaluer la vitesse à laquelle le Tangle se construit et donc la fréquence à laquelle on approuve les transactions. Il est important de noter qu'on se réfère non pas au temps de la simulation mais à son temps d'exécution en un autre terme le **CPU time**. Cette métrique va donc dépendre de la puissance de la machine simulant le Tangle.

— **Le nombre de transactions approuvées** : ce critère permet d'établir l'efficacité d'un TSA à approuver un nombre satisfaisant de transactions.

— Et finalement, d'une manière analogue, **le nombre de tips**.

Tout comme pour la visualisation, nous utiliserons des fichiers logs afin de pouvoir mesurer ces différentes métriques. Concernant le temps d'exécution, on utilisera l'interface **Cmdenv** fourni par OMNeT++ et le mode **Express-Mode** permettant d'obtenir dans des fichiers logs des informations sur la progression et la performance de la simulation :

```
Running configuration General, run #0...
Assigned runID=General-0-20210522-16:26:41-17676
Setting up network "TangleSim"...
Initializing...

Running simulation...
** Event #0   t=0   Elapsed: 2e-006s (0m 00s)
     Speed:     ev/sec=0   simsec/sec=0   ev/simsec=0
     Messages:  created: 10   present: 10   in FES: 10
** Event #1536   t=403.062   Elapsed: 2.86421s (0m 02s)
     Speed:     ev/sec=536.624   simsec/sec=140.724   ev/simsec=3.81331
     Messages:  created: 1552   present: 16   in FES: 15
```

FIGURE 7 – Output du mode Express-Mode de Cmdenv

Pour extraire le temps d'exécution du fichier, on utilisera le script `Stats.py` et des **regex**. Bien plus, ce script permet aussi d'effectuer une moyenne s'il y a existence de plusieurs fichiers logs. Finalement, avant qu'une simulation termine, on copiera le nombre de tips dans un ficher txt nommé NumberTips.txt.

6 Les différents TSA

6.1 IOTA

Le premier algorithme de sélection que nous allons étudier est celui du papier original [6]. Le principe est de placer N particules qu'on nommera walkers (i.e « marcheurs ») sur des sites du Tangle de manière totalement aléatoire. Chacun de ces walkers va ainsi initier une marche aléatoire pour atteindre un tip. Finalement, les deux tips atteints en premier seront approuvés. Bien plus, les walkers seront placés dans un intervalle $[W, 2W]$ où W est un entier, cela signifie que les N particules devront être placées entre le W_{ieme} site et le $2 \times W_{ieme}$. Pour sélectionner sur quels sites aller lors de la marche aléatoire, on se référera à la probabilité de passage entre un site x et y dans le papier [6], page 21 :

$$P_{xy} = \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_y)\big)\Big(\sum_{z:z\rightsquigarrow x} \exp\big(-\alpha(\mathcal{H}_x - \mathcal{H}_z)\big)\Big)^{-1}$$

avec $\alpha \geq 0$ et $\mathcal{H}_x$ le poids cumulé du site x.

Algorithm 1: Marche aléatoire du papier [6] utilisant une MCMC

Input : - N, le nombre de walkers
- W, un entier
- α, un réel positif

Output: Deux tips à approuver

1 **Pseudo-code :**

1. Initier indépendamment N marches alétoires dans un intervalle $[W, 2W]$,
2. Simuler à chaque marche la probabilité P_{xy} pour le passage d'un site x à un site y,
3. Lorsqu'un tip est atteint, il est stocké et la marche alétoire s'arrête.
4. Finalement, lorsque les N marches alétoires ont terminé, sélectionner les deux tips atteints en premier comme ceux à approuver.

Il important de noter que dans notre implémentation, le poids propre de chacune des transactions sera initialisé à **1**. De plus, le critère de rapidité entre deux walkers sera le nombre de **marches effectuées** avant d'atteindre un tip.

Le paramètre α permet de caractériser la marche aléatoire : déterministe ou uniforme. En d'autres termes, si celui-ci est très élevé alors l'influence du poids cumulé dans la probabilité de passage d'un site x vers un site y devient importante ce qui signifie qu'on aura beaucoup plus de chance d'aller vers un site ayant un poids cumulé conséquent. A l'inverse, lorsqu'α est très faible, on obtient une marche aléatoire uniforme ce qui signifie que la probabilité de passage vers chacun des sites voisins est identique. Ainsi, le choix du paramètre α est très important car comme on le verra, celui-ci permet de modifier totalement la forme du Tangle. En effet, en ayant une marche aléatoire très déterministe, celle-ci sélectionnera uniquement les tips approuvant indirectement les sites ayant les poids cumulés les plus élevés. Par conséquent, cela pose un énorme problème car nous aurons que très peu de tips approuvés, et donc de transactions.

Par exemple, voici deux Tangles construits avec deux α différents :

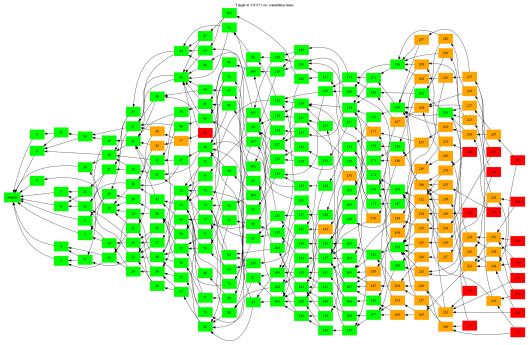

FIGURE 8 – Tangle construit avec $\alpha = 0.2$

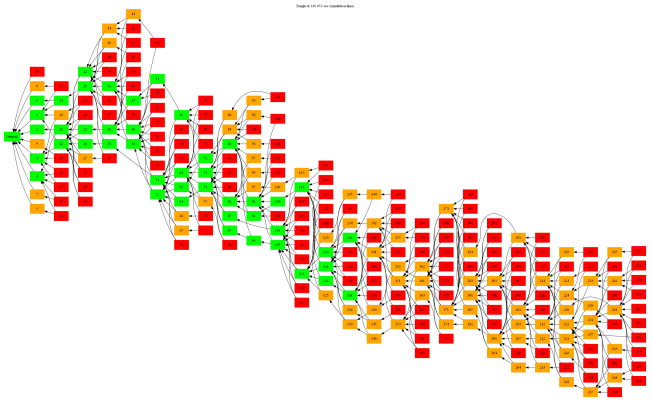

FIGURE 9 – Tangle construit avec $\alpha = 0.8$

Nous remarquons clairement que pour un α très faible, nous avons beaucoup plus de transactions approuvées que pour un α élevé. Nous pouvons donc exclure l'utilisation d'un α élevé. Cependant, l'utilisation d'un alpha très faible fragilise la sécurité du Tangle (voir le papier [6] partie 4, page 15). Il est donc primordial d'utiliser un α élevé afin de garantir la sécurité du Tangle. Par conséquent, on ne peut à la fois garantir la sécurité du Tangle et permettre la validation d'un nombre satisfaisant de transactions avec cet algorithme.

Pour remédier à cette contrainte, le papier [2] propose **G-IOTA**.

6.2 G-IOTA

Le papier [2] propose une différente approche pour l'implémentation de la marche aléatoire, nommée G-IOTA. Celle-ci se repose sur le principe d'approuver non pas deux tips mais trois afin d'accorder une seconde chance aux tips qui n'ont pas été approuvés auparavant. Pour cela, on sélectionnera de façon aléatoire un troisième tip à approuver [2] :

Algorithm 2: G-IOTA [2]

Input : - N, le nombre de walkers
- W, un entier
- α, un réel positif

Output: Trois tips à approuver

1 **Pseudo-code :**

1. Exécuter l'algorithme 1 avec les paramètres N, W et α,
2. Choisir aléatoirement un troisième tip différent à approuver.

On remarquera que G-IOTA utilise la marche aléatoire du papier original pour obtenir les deux premiers tips. Ainsi, il est primordial d'utiliser G-IOTA avec un α élevé pour garantir la sécurité du Tangle, quant à la validation d'un montant satisfaisant de transactions, cette condition sera assurée par la validation d'un troisième tip.

Bien plus, il est intéressant de noter que tout comme la marche aléatoire du papier [6], si nous n'arrivons pas à obtenir le nombre nécessaire de tips à approuver (par exemple lorsque le Tangle n'a pas assez de tips, ce qui est le cas lorsqu'on est à t (temps) petit), il ne faut pas sélectionner plus d'une fois un tip. En effet, il est préférable d'avoir des tips différents à approuver et non des doublons, quitte à ne pas satisfaire le nombre de tips à approuver.

2. Cette approche peut différer, voir le papier [2] partie III.B

Voici le Tangle qu'on obtient avec cette implémentation :

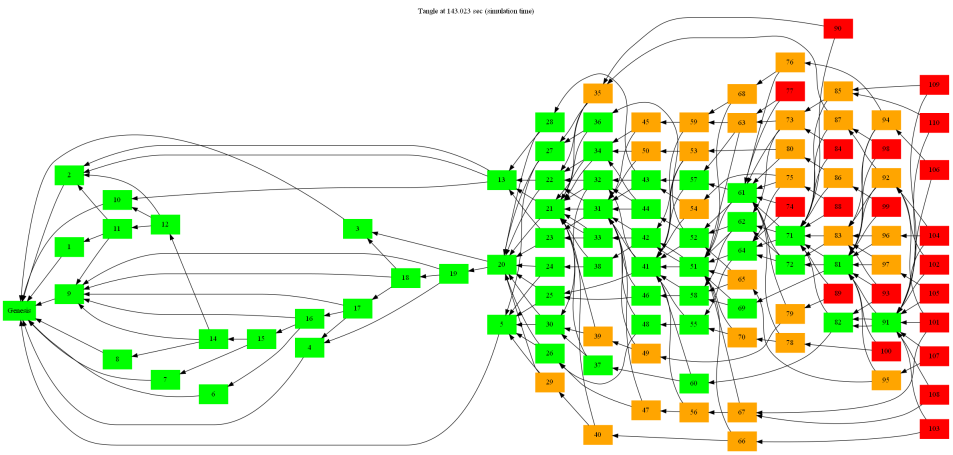

FIGURE 10 – Tangle construit avec G-IOTA

On remarque que nous avons énormément de transactions approuvées malgré le fait qu'on utilise un alpha très grand, garantissant ainsi la sécurité du Tangle.

Cependant, cette implémentation possède des limites :

— Valider un troisième tips, signifie effectuer une troisième proof-of-work, impliquant donc du calcul supplémentaire qui peut se répercuter sur un ralentissement du réseau si nous n'avons pas de puissance de calcul supplémentaire.
— De même, le choix aléatoire d'un troisième tip demande des ressources de calcul supplémentaires.
— Finalement, la marche aléatoire demeure déterministe c'est-à-dire qu'on peut anticiper à chaque marche, le site sur lequel le walker ira, ce qui implique qu'on n'a plus de hasard. Cela affaiblit donc la sécurité du Tangle car on peut anticiper la marche aléatoire et connaître par avance le tips qui a va être sélectionné.

Pour corriger ces différents problèmes, le papier [3] propose **E-IOTA**, une autre implémentation de la marche aléatoire.

6.3 E-IOTA

Tout comme G-IOTA, E-IOTA utilise l'algorithme de marche aléatoire proposé par le papier [6]. Cependant, ici nous allons générer un nombre aléatoire r à chaque fois qu'une transaction est émise tel que $r \in [0, 1)$ et poser $p1/p2$ tels que $0 < p1 < p2 < 1$. Ainsi :

Algorithm 3: E-IOTA [3]

Input : - N, le nombre de walkers
- W, un entier
- $r \in [0, 1]$, un réel généré aléatoirement
- $p1/p2 \in]0, 1[$

Output: Deux tips à approuver

Pseudo-code :

if $r < p1$ **then**
    on effectue une marche aléatoire uniforme, i.e, avec $\alpha = 0$ en utilisant les paramètres N et W.
else if $p1 \leq r < p2$ **then**
    on effectue une marche aléatoire avec un α faible en utilisant les paramètres N et W.
else if $p2 \leq r < 1$ **then**
    on effectue une marche aléatoire avec un α élevé en utilisant les paramètres N et W.

Voici le Tangle que nous obtenons avec E-IOTA :

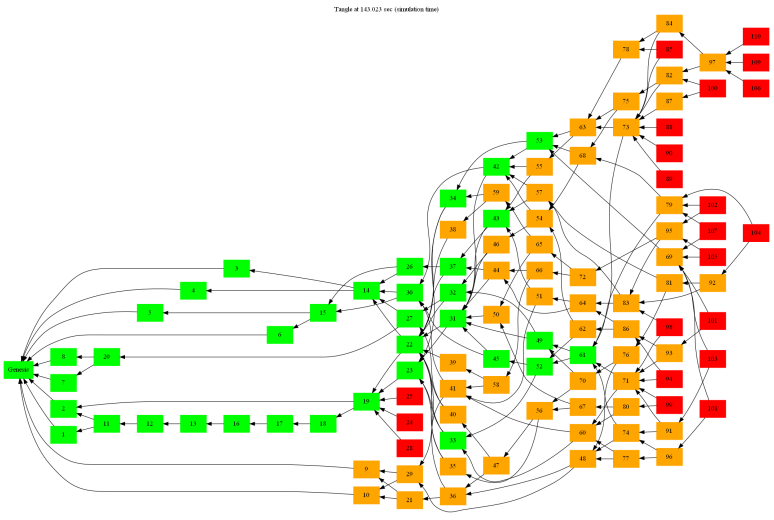

FIGURE 11 – Tangle construit avec E-IOTA

7 Résultats

Dans cette partie nous allons discuter et comparer les différents TSA évoqués auparavant selon les métriques présentées à la partie 5. Pour cela, nous allons simuler **100 Tangle** d'une taille de **1000 transactions** en utilisant à chaque fois l'un des TSA puis nous effectuerons une **moyenne** des résultats obtenus avec le script `Stats.py`.

Concernant les spécifications et les paramètres des TSA, de la simulation et du système, les voici :

Paramètres simulation :
- Nombre de tests (paramètre repeat dans le fichier INI) : 100
- Interface : Cmdenv
- Mode : Express-Mode
- PoW time : 10.0 sec
- Nombre de nœuds (utilisateurs du réseau) : 10
- Fréquence d'émission d'une transaction : 3.0 sec
- Nombre de transactions simulées : 1000

Paramètres système :
- CPU : AMD R5 1600 @3.8 Ghz
- RAM : 16 GB
- OS : Windows 10 ver. 2004
- Logiciel : OMNeT++ ver. 5.6.2

IOTA :
- $N = 10$
- $W = 250$
- $\alpha = 5.0$

G-IOTA :
- $N = 10$
- $W = 250$
- $\alpha = 5.0$

E-IOTA :
- $N = 10$
- $W = 250$
- $p1 = 0.1$
- $p2 = 0.65$
- $\alpha_{low} = 0.1$
- $\alpha_{high} = 5.0$

Les résultats sont présentés sous forme d'un barplot :

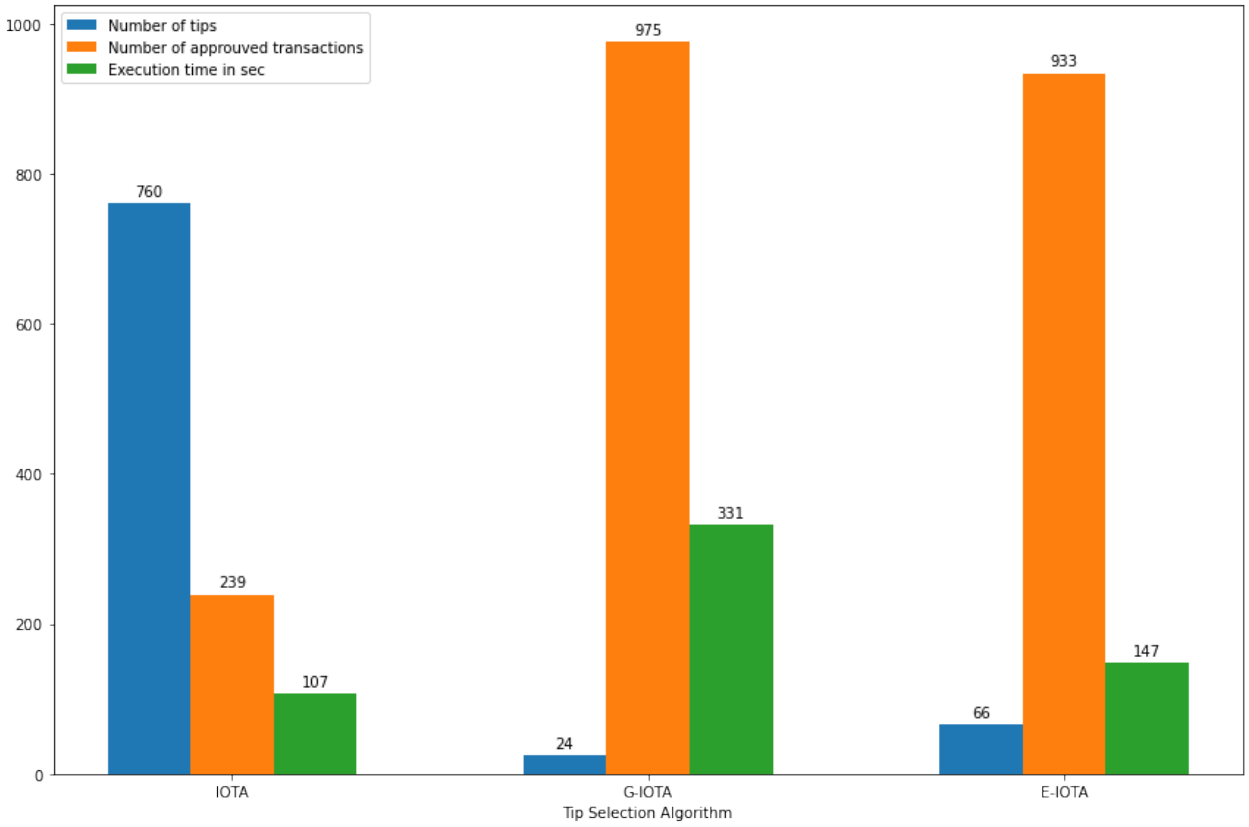

FIGURE 12 – Comparaison entre les différents TSA

Nous avons, en bleu, le nombre de tips pour chaque TSA utilisé. On remarque qu'IOTA possède en moyenne le plus de tips, avec 76% des transactions émises étant des tips. Par analogie, cela se reflète sur le nombre de transactions validées avec un pourcentage de validation faible de 24%. Concernant E-IOTA et G-IOTA, nous avons des résultats très similaires avec un taux de validation supérieur à 90%.

En vert, nous avons le temps d'exécution mis par chaque TSA afin de construire le Tangle. Comme dit précédemment, G-IOTA est le plus gourmand en ressources calculatoires, il est donc normal qu'il soit le plus lent à s'exécuter. Au niveau d'IOTA et E-IOTA, nous avons des résultats très similaires avec un temps d'exécution avoisinant les ~ 2 min.

Finalement, d'un point de vue général, ces barplots montrent bien que E-IOTA est le TSA ayant le meilleur compromis.

8 Conclusion

Pour ce projet, notre objectif était, en premier lieu, de comprendre le fonctionnement de IOTA par la lecture et l'assimilation de papier de recherche [6]. Notre seconde étape fut de comprendre, assimiler et vérifier le code du git [5] visant à simuler le Tangle. Cela a impliqué de nous familiariser avec l'environnement OMNeT++ et le C++ pour pouvoir comprendre son implémentation. Nous avons ensuite ajouté des moyens de visualisation et de tracking afin d'avoir un réel suivi de la construction du Tangle. Une fois ces étapes accomplies, nous pouvions enfin commencer à l'implémentation des TSA. Pour cela nous nous sommes appuyés sur les papiers [6], [2] et [3], avec les deux derniers offrant des pistes d'améliorations de la marche aléatoire du premier papier.

Finalement, nous avons mis en place une méthode d'évaluation afin de comparer les performances de chacun des TSA :

— Premièrement, nous obtenons des résultats qui vont de pair avec les informations discutées dans la partie 6. En effet, on remarque que le TSA proposé par IOTA ne peut être implémenté avec un α élevé au risque d'avoir un taux de validation très faible. Bien plus, on remarque que G-IOTA n'est pas un TSA rapide.
— Deuxièment, nous obtenons des résultats très similaires au papier [3]. Nous avons tout de même une différence minime en termes de temps d'exécution entre IOTA/E-IOTA et du taux de validation entre G-IOTA/E-IOTA. Cela s'explique pour le premier cas par le calcul récursif du poids de tous les sites malgré un α nul (lorsque que nous avons $r < p1$) et pour le second cas, par la différence de notre implémentation de G-IOTA concernant le choix du troisième tip (voir 2).
— Finalement, on peut conclure qu'E-IOTA est un bon candidat pour remplacer le TSA d'IOTA car il utilise la même marche aléatoire basée sur une MCMC décrite dans le papier [6], qu'il a un taux de validation très satisfaisant (de 93%) mais aussi un temps d'exécution très court. Tout cela en garantissant la sécurité du Tangle théoriquement, comme décrit par le papier [3].

La prochaine étape est de mettre en place des attaques visant le Tangle afin de vérifier les informations avancées par les papiers [2] et [3] en terme de sécurité. Également, nous avons comme ambition de décentraliser notre simulation.

9 Impacts

Les cryptomonnaies sont d'actualité et auront de nombreux impacts aussi bien environnementaux, éthiques que sociétaux. Concernant les impacts environnementaux, ils sont négatifs puisque ces crypto-monnaies ont besoin de serveurs/mineurs pour fonctionner qui sont très coûteux en énergie. Par exemple, on est de l'ordre de **115 TWh par an** pour le Bitcoin, ce qui le placerait en **33ème pays** le plus consommateur d'énergie entre les Pays-Bas et les Emirats Arabe Unis :

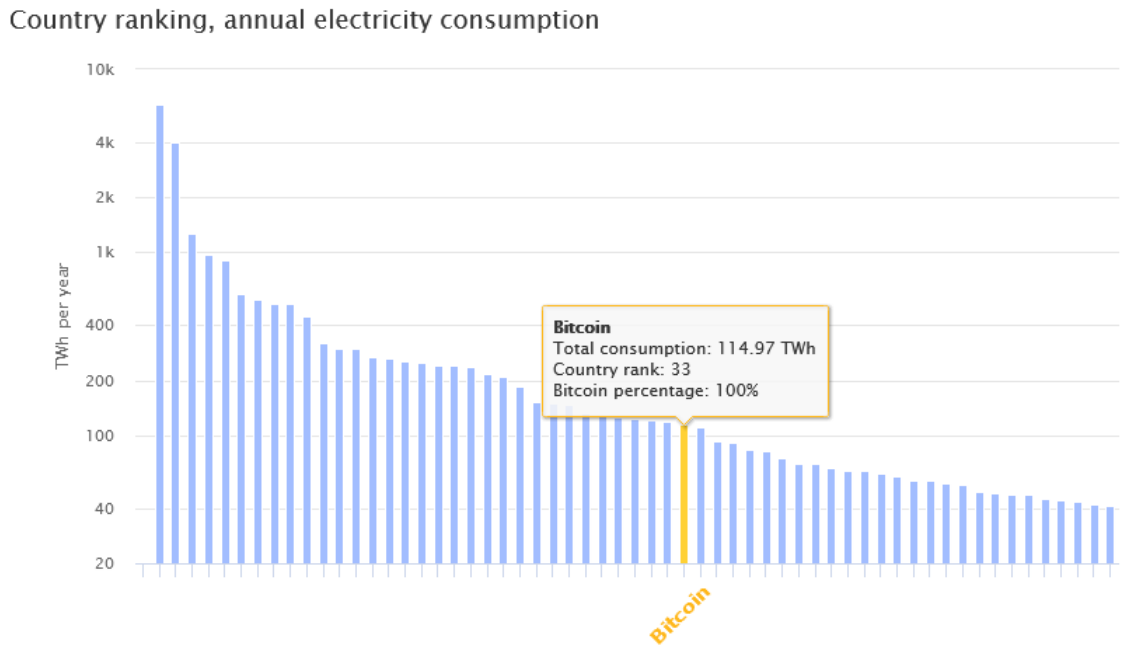

FIGURE 13 – Consommation annuelle du BTC, source : [4]

Ainsi, la popularisation des cryptomonnaies posera donc un sérieux problème énergétique et donc environnemental.

D'un point de vue sociétal, les crypto-monnaies peuvent et ont déjà modifié les moyens de paiements traditionnels et donc bousculées une grande partie de notre société (voir l'offensive du gouvernement chinois contre la production et le trading du Bitcoin par exemple). Bien plus, les applications relatives aux crytomonnaies ne s'arrêtent pas au monde de la finance mais peuvent affecter comme nous l'avons vu avec IOTA des domaines comme de l'IoT, qui aujourd'hui est présent tout autour de nous.

10 Bilan personnel

Léo : J'ai trouvé le sujet très intéressant et enrichissant du point de vue des connaissances acquises. Au début du projet nous ne connaissions que très peu l'univers de la crypto-monnaie, nous n'avions jamais entendu parler du logiciel OMNet++ et ne savions pas coder en C++. Ce projet m'a apporté des connaissances importantes qui me serviront.

Tahar : Ce projet m'a permis de découvrir et d'apprendre un nouveau langage qu'est le C++. En effet, étant habitué au C et à Python, j'ai dû sortir de ma zone de confort. J'ai d'ailleurs découvert un nouveau logiciel qu'est OMNeT++, logiciel très complet qui pourra peut-être me servir plus tard, mais aussi avec surprise l'apport scientifique important derrière le monde des cryptomonnaies. Bien plus, la reprise d'un projet écrit par une autre personne n'est pas une compétence simple à développer/acquérir, cependant, ce projet m'a permis de le faire. Finalement, je suis impatient de continuer ce projet durant mon stage cet été afin de finir ce qu'il nous restait à effectuer.

Emile : Travailler sur ce projet m'a permis d'en apprendre plus sur les blockchains et donc de discerner le vrai du faux sur les nombreuses informations que l'on peut voir sur ce sujet. Mais ce projet m'a aussi permis d'apprendre à utiliser OMNet++ qui est un logiciel très utilisé en réseau et difficile à prendre en main donc je pense que cela me sera utile plus tard que ce soit dans ma vie étudiante ou professionnelle. De plus, nous avons travaillé en C++ ce que je n'avais jamais réalisé auparavant, j'ai donc pu apprendre les bases de ce langage. Finalement ce projet a évidemment amélioré mes capacités à travailler en équipe que ce soit du côté coordinations lorsque nous devions nous répartir des tâches et travailler séparément ou du côté adaptations lorsque nous avions à travailler ensemble et devions se mettre d'accord.

Ghassen : Grâce à ce projet, j'ai pu améliorer mes connaissances sur les crypto-monnaies et sur les techniques utilisées pour valider des transactions (blockchain, Tangle ...). Particulièrement, on a travaillé sur la démarche pour valider des transactions dans IOTA qui est une crypto-monnaie émergente. J'ai dû m'initialiser au language C++ pour prendre en main OMNeT++ sur lequel les simulations du Tangle sont faites.

11 Bibliographie

[1] Tahar Amairi, Léo Lassalle, Emile Pichard et Ghassen Hachani. *Implementation of different TSA for IOTA using OMNeT++*. URL : https://github.com/T-amairi/Implementation-of-different-TSA-for-IOTA.

[2] Gewu Bu, Önder Gürcan et Maria Potop-Butucaru. *G-IOTA: Fair and confidence aware tangle*. URL : https://ieeexplore.ieee.org/document/8845163.

[3] Gewu Bu, Wassim Hana et Maria Potop-Butucaru. *E-IOTA: an efficient and fast metamorphism for IOTA*. URL : https://ieeexplore.ieee.org/document/9223294.

[4] University of Cambridge. *Cambridge Bitcoin Electricity Consumption Index*. URL : https://cbeci.org/.

[5] Richard Gardner. *Omnet++ simulation of a Tangle*. URL : https://github.com/richardg93/TangleSim.

[6] Serguei Popov. *The Tangle*. URL : http://www.descryptions.com/Iota.pdf.